NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

INTRODUCTION TO HIDDEN MARKOV MODELS AND THEIR APPLICATIONS TO CLASSIFICATION PROBLEMS

by

Michail Zambartas

September 1999

Thesis Advisor: Monique P. Fargues
Co-Advisor: Roberto Cristi

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1999	Master's Thesis

4. TITLE AND SUBTITLE
INTRODUCTION TO HIDDEN MARKOV MODELS AND THEIR APPLICATIONS TO CLASSIFICATION PROBLEMS

5. FUNDING NUMBERS

6. AUTHOR(S)
Michail Zambartas

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

This thesis presents an introduction to Hidden Markov models (HMM) and their applications to classification problems. HMMs have been used extensively to model the temporal structure and variability of speech and other signals in the last decade. We selected to write our own HMM implementation in MATLAB. We tested our software on a limited isolated 4-word recognition. We also applied our implementation to the recognition of mine-like objects buried in shallow sand, using seismo-acoustic data obtained from an on-going project at the Naval Postgraduate School. Initial results indicate that the HMM-based classifier can recognize the type of mine-like object, independent of the object weight with a 97% accuracy. Results also indicate that it can recognize the object type at different distances with a 100% accuracy. However, the experiments were conducted with very few data, and further work needs to be done to confirm these initial findings by using a larger data set. Finally, we benchmarked our results against those obtained using a back-propagation neural network implementation, which were found to be similar, but slower than the HMM-based implementation.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Hidden Markov models, vector quantization, speech recognition, seismo-acoustic sonar, mine detection	153
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

StandardForm298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18-102

Approved for public release; distribution is unlimited

INTRODUCTION TO HIDDEN MARKOV MODELS AND THEIR APPLICATIONS TO CLASSIFICATION PROBLEMS

Michail Zambartas
Lieutenant, Hellenic Navy
B.S., Hellenic Naval Academy, 1990

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 1999

ABSTRACT

This thesis presents an introduction to Hidden Markov models (HMM) and their applications to classification problems. HMMs have been used extensively to model the temporal structure and variability of speech and other signals in the last decade. We selected to write our own HMM implementation in MATLAB. We tested our software on a limited isolated 4-word recognition. We also applied our implementation to the recognition of mine-like objects buried in shallow sand, using seismo-acoustic data obtained from an on-going project at the Naval Postgraduate School. Initial results indicate that the HMM-based classifier can recognize the type of mine-like object, independent of the object weight with a 97% accuracy. Results also indicate that it can recognize the object type at different distances with a 100% accuracy. However, the experiments were conducted with very few data, and further work needs to be done to confirm these initial findings by using a larger data set. Finally, we benchmarked our results against those obtained using a back-propagation neural network implementation, which were found to be similar, but slower than the HMM-based implementation.

TABLE OF CONTENTS

I. INTRODUCTION

This thesis presents an introduction to Hidden Markov models (HMM) and their applications to classification problems. HMMs have been used extensively to model the temporal structure and variability of speech and other signals in the last decade. For example, they have become a major player in the speech area [4, 5, 9]. We selected to write our own HMM implementation even though sophisticated HMM software is readily available on the market to better understand the basic concepts behind the theory. Our implementation uses MATLAB because it is a high-level language easy to work with. As a result, the software may not be as fast as that obtained with a compiled implementation but it is easy to understand, which was one of the main goals of the research. We tested our software on a limited isolated 4-word recognition, and we also applied our implementation to the recognition of mine-like objects buried in shallow sand, using seismo-acoustic data obtained from an on-going project headed by Prof. Muir from the Physics Department at the Naval Postgraduate School. Finally, we benchmarked our results against those obtained using a back-propagation neural network implementation.

Chapter II introduces the concepts of HMMs in an "engineering-oriented" simple fashion. We illustrate the theoretical concepts with basic examples to facilitate the understanding of this difficult topic. We cover the three specific problems which HMM address, their solutions, emphasizing the computational savings obtained with the forward, backward, Viterbi and Baum-Welch algorithms.

Chapter III presents the application of HMMs to a simple speech classification problem, using four isolated three-syllable words: "Microsoft," "Statistics," "Instructor" and "Professor." Our goal is to show how a generic classifier can be set-up through the simple speech recognition example. We introduce the concept of vector quantization (VQ) applied to generate discrete symbols from the speech feature vectors created with Linear Prediction Coefficients (LPC) and energy coefficients. Two different implementations of VQ are considered 1) a Neural Network (NN)-based VQ scheme; and 2) a K-means algorithm [9]. In addition, we point out the potential numerical difficulties which can be encountered while setting up and implementing the HMM software.

Chapter IV considers the application of the HMM-based classifier to the classification of two mine-like objects buried in sand; a cylinder and a powder keg with weights ranging from 71kg to 290kg. Results show that recognition performance is good under various conditions of weight and distance of the object.

Chapter V presents a back-propagation neural network classifier designed to recognize the two mine-like objects discussed in Chapter IV. This implementation was conducted on the same data to compare the performance of the two schemes. Results show the performances to be similar (around 95%), but the HMM-based implementation procedure is faster than the NN-based implementation.

II. INTRODUCTION TO HIDDEN MARKOV MODELS

This chapter introduces the basic theory of Hidden Markov models (HMM), which are a very powerful technique for modeling the temporal structure and variability in speech and other applications. The understanding of HMM is very important in order to use them properly and achieve the best results for signal classification.

A. HMM BACKGROUND

Hidden Markov Model theory was first introduced in the late 1960s and early 1970s by Baker at Carnegie-Mellon University and Jeninek and colleagues at IBM for speech recognition [6]. Since then they have been used extensively for speech applications, and also successfully to other tasks such as human face identification, lip and speech-reading, optical character recognition and time DNA modeling ([6] and references therein). The reason for this wide range of applications is the rich mathematical structure the HMMs are built on, yielding optimal results if used properly. A detailed overview of HMM theory was presented by Rabiner in the late 1980s [3, 4].

There are three main reasons why we may need to model a signal. First, we apply signal modeling to mathematically describe the signal, so that we'll be able to process it, for example to denoise a speech signal. Models are also important because they let us describe the signal source, which doesn't have to be directly available to the user. For example we cannot produce real seismic waves without an earthquake, but we can create models of such signals and then process them. Finally, the most important reason for the widely spread use of signal models is that they perform well in practice [3].

There are two types of signal models; deterministic and statistical. Deterministic modeling is applied when dealing with signals with known physical characteristics. For

example a sinewave is completely specified by its frequency, phase and amplitude. Stochastic modeling is applied when one tries to characterize only the statistical properties of the signal. For example, statistical modeling may include Gaussian Poisson and Markov process to describe events. Usually, real applications use both deterministic and stochastic modeling. In this work we focus on statistical signal modeling, using the Hidden Markov Model (HMM).

B. INTRODUCTION

The HMM theory is based on the Markov Chain. We can define the Markov Chain as a probabilistic description of transitions between a system's states. A state can be a property, or generally a condition, that a system/model might have at a particular instance. The HMM consists of an underlyining Markov chain describing the probabilistic status between the states, as that shown in Figure 2.1, which illustrates a three state left-right model. For example, suppose we want to model a speech signal. First, the signal is split into T time frames. Then, a set of parameters (such as LPC coefficients, energy, etc...) is extracted together with a set of symbols for each time frame. The sets of symbols represent each frame characteristics, and are called observations. As a result, the entire model is a sequence of symbols, and each symbol is a system model depicting each segment. In most cases we choose the segment length empirically, but sometimes adjust it so that it is large enough to contain all the information (usually spectral) that makes it unique, by comparison with the other segments, due to possible change in the signal behavior. Generally, we can assume that we reach an optimal number of frames when, by decreasing the frame length, we generate models identical to those already generated. At this point, HMMs take advantage of the

properties existing between adjacent segments properties by addressing the following three problems [3]: 1) how to identify the characteristic frames; 2) how to characterize the relation between all successive segments; and 3) what types of properties should be extracted to model each segment.

C. DISCRETE MARKOV PROCESS

An accurate definition for the HMM according to Rabiner [3] is that "A HMM is a doubly stochastic process that is not observable (it is hidden), but can only be observed through another set of stochastic processes that produce the sequence of observed symbols."

Consider a system which exists at time t in one of N possible potential states, as illustrated in Figures 2.1 and 2.2, where N=3. Each of the three circles represents a state of the model. At a specific discrete time instant t within frame k, the model is always at one of those three states, and we observe the sequence O_k. Generally, given that the system has N states $S = \{s_1, s_2, \ldots, s_N\}$, where s_j is the j^{th} state, at every time t=k, the model passes through the sequences of states $Q = \{q_1, q_2, \ldots, q_t\}$, where q_k is the state at time k. The model that describes all this information is called a Markov chain. As we can see from the example in Figure 2.1,

$$\exists k : q_k = q_{k+1}, \tag{2.1}$$

which means that if the model at time t=k is at state, it can remain at the same state at time t=k+1. Note that no backward transitions are allowed in Fig 2.1. As a result, this model is called a "left-to-right" model (the model is called ergotic when backward transitions are allowed, as illustrated in Fig 2.2).

We define the probability a_{ij} as the transition probability from state i to state j. For example, a_{23} is the probability of going from the 2nd to the 3rd state, and is defined as:

$$a_{ij} = P[q_t = j | q_{t-1} = i], \; 1 \le i,j \le N, \tag{2.2}$$

with the following constrains:

$$\begin{aligned} a_{ij} &\ge 0 \qquad \forall i, j, \\ \sum_{j=1}^{N} a_{ij} &= 1 \quad \forall i, \end{aligned} \tag{2.3}$$

since all probabilities are positive numbers and the summation of all transition probabilities is 1.

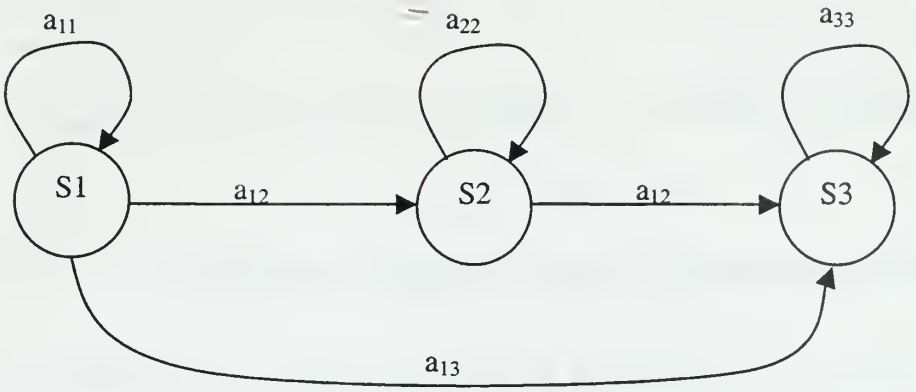


Figure 2.1 A Markov Process (Chain), left-to-right model

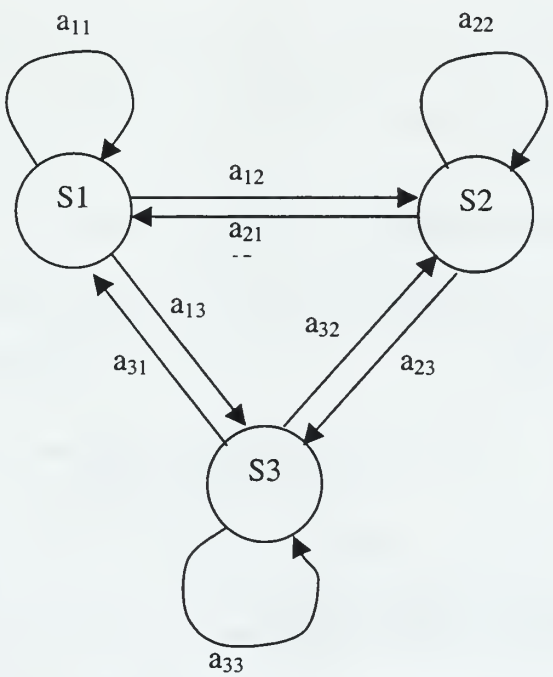


Figure 2.2 A Markov Process (Chain), ergotic model

For the discrete-time case, a system governed by known or predictable dynamics can be modeled using a Markov chain. The probability of observing the observation O_t, given the model λ at a time t, is determined by the state at the time preceding states [6]:

$$P(O_t|\lambda) = P[q_t = s_j, q_{t-1} = s_i, q_{t-2} = s_k, \ldots]. \tag{2.4}$$

For a first-order Markov chain, eq (2.4) can be simplified to:

$$P(O_t|\lambda) = \prod_{k=1}^{t} P[s_k \mid s_{k-1}] \cdot P[s_0], \qquad 1 \le t \le T. \tag{2.5}$$

Finally, we define the initial conditions, the probabilities of starting (t=1) at the i^{th} state s_i as:

$$\pi_i = P[q_1 = s_i], \qquad i = 1, 2, \cdots N. \tag{2.6}$$

1. Example 1

Let's evaluate the probability of the observation $O = \{s_1, s_1, s_3\}$ for the example shown in Figure 2.2. So, applying eqs (2.4) and (2.5) to the example shown in Figure 2.1 leads to:

$$\Pr(O|\lambda) = P[\{s_1\ s_1,\ s_3\}|\lambda]$$
$$\Pr(O|\lambda) = P[s_3|s_2]P[s_2|s_1]P[s_1]$$
$$\Pr(O|\lambda) = \pi_1 a_{23} a_{12}.$$

2. Example 2: Browser Tracking.

Next, let us consider the following example where a user wishes to track the browser type used by visitors to the Web. Assume: 1) only three types of browsers are available: Microsoft Explorer (MIE), Netscape (NET) and America Online' s browser (AOL); 2) the specific browser type version is not taken into account.

Table 2.1 Browser Tracking

User #	1	2	3	4
Browser	MS Explorer5.0	AOL4.0	Netscape3.0	Netscape6.1
Browser Observation Symbol	*1*	*2*	*3*	*3*

An observation symbol is assigned to each visitor's browser types, as illustrated in Table 2.1. Note that we assign the same symbol (3) to both versions of Netscape, as the specific browser version in not tracked by the user. In addition we assume transition probabilities which represent probabilities of going from one state to another, i.e., browser, to another are available from previous statistical studies:

$$a = \begin{bmatrix} 0.3 & 0.4 & 0.3 \\ 0.2 & 0.5 & 0.3 \\ 0.8 & 0.1 & 0.1 \end{bmatrix}.$$

Thus, the resulting observation sequence is O={1, 2, 3, 3}. Finally, let's assume that the probability the first user uses the explorer browser is π_1=0.5.

Therefore:

$$P(O|\lambda) = P[1, 2, 3, 3|\lambda] = P[1]P[2|1]P[3|2]P[3|3] = \pi_1 a_{21} a_{32} a_{33}$$
$$\Rightarrow P(O|\lambda) = 0.5 \cdot 0.2 \cdot 0.1 \cdot 0.1 = 0.001.$$

3. Example 3: Urn-ball Example

Note that the states are directly observable in the previous examples. However, in most real world cases, the state sequence that produces a given sequence of patterns cannot be determinated, and the model is said to be "hidden". The most common example for this type of description is the urn-ball model, according to Fig 2.3:

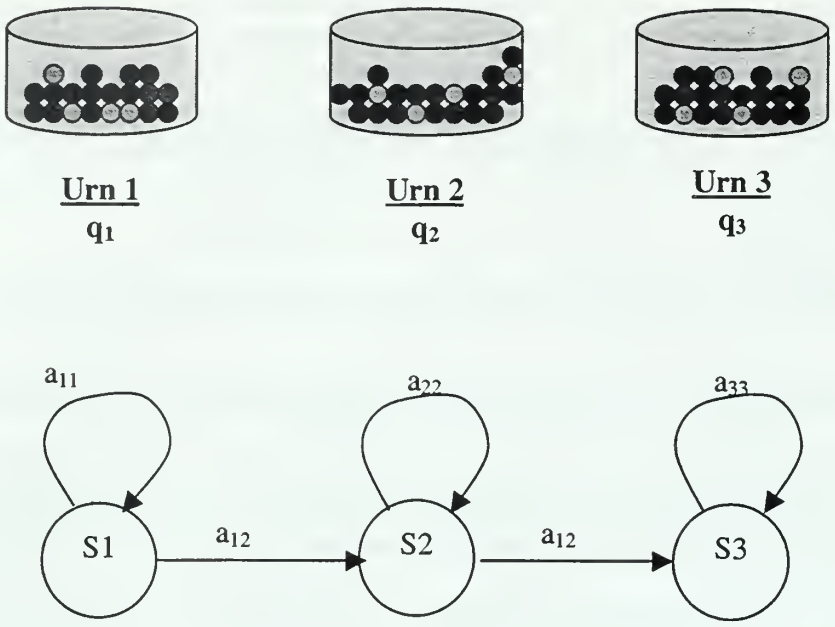


Figure 2.3. The classic Urn-ball example. There are three (N=3) urns that contain a large number of color balls. The colors of the balls are red, green, and blue. The boy we assigned to paint the balls runs out of blue paint sometime while painting and uses sky blue afterwards. Thus, eventhough there are 4 colors, we assign the same symbol for blue and sky blue and the number of symbols is three (M=3). Each time a person randomly selects an urn, he selects a ball, and announces the color. The process is repeated 4 (T=4) times. Note that: 1) we don't know from which urn the ball came from, as we only have a color sequence, and we map those colors with the symbols according from Table, 2.2; 2) we assume that it is a left-to-right model for simplicity.

Table 2.2. Urn-ball example. T, Q, C, O values parameters

Clock Time T	1	2	3	4
Urn (Hidden State) Q	q_1	q_1	q_2	q_3
Color C	Red	Blue	Sky Blue	Green
Observation Symbol O_s	1	2	2	3

Table 2.2 shows that we have T=4 observations of four balls. The number of urns is N=3 and each urn represents one of the 3 hidden states, because we don't know from

the specific urn number each ball comes from. Further, we consider that the sky blue color belongs to the same class as blue, so we assign the same observation symbol (O_s=2) for both colors, and the total number of symbols is M=3. The possible observation symbols are V={1, 2, 3}, and the states are Q = {q_1, q_2, q_3}. We also assign values to the transition probability matrix A={a_{ij}}, and initial conditions vector Π={π_i}:

$$A = \{a_{ij}\} = \begin{bmatrix} 0.5 & 0.3 & 0.2 \\ 0 & 0.8 & 0.2 \\ 0 & 0 & 1 \end{bmatrix}, \qquad \Pi = \{\pi_i\} = \{0.9,\ 0.1,\ 0\}.$$

Note that the specific upper triangular structure of A indicates the model considered here is left-to-right, as we cannot go backward. In addition, we assume that the probability of selecting the first urn is 90%. Finally, we also need the observation symbol probability distribution at every state, to describe the model completely. This information is presented in the matrix B, which contains the probabilities of being at the state j and observing the symbol k, such as:

$$B = \{b_{jk}\}, \qquad 1 \le j \le N,\ 1 \le k \le M\ , \tag{2.7}$$

with the following properties (like A):

$$\begin{aligned} & b_{jk} \ge 0 \qquad \forall j,k\ , \\ & \sum_{k=1}^{M} b_{jk} = 1 \quad \forall j. \end{aligned} \tag{2.8}$$

Note that B is a NxM matrix where N represents the number of hidden states and M the number of symbols. Matrix B isn't restricted to be square. In our example, a possible B matrix may be defined as:

$$B = b_{jk} = \begin{bmatrix} 0.3 & 0.5 & 0.2 \\ 0.4 & 0.4 & 0.2 \\ 0.6 & 0.3 & 0.1 \end{bmatrix}.$$

The statistical model is completely described by the set of matrices A, B, and the vector π and usually denoted as:

$$\lambda = \{A, B, \pi\}. \tag{2.9}$$

Note that all rows of matrices A, B and π shown sum-up to 1, thereby providing an easy check on the validity of the model. Extending our example to a general case, Table 2.3 lists the definition of all HMM parameters, as given by Rabiner [3], for a generic HMM model:

Table 2.3 HMMs elements

- **T: Length of the Observation Sequence (total number of clock times)**
- **N: Number of States in the Model**
- **M: Number of observation symbols**
- **$Q=\{q_1, q_2, \ldots, q_N\}$: States**
- **$V=\{v_1, v_2, \ldots, v_M\}$: Discrete Set of Possible Symbol Observations**
- **$A = \{a_{i,j}\}$, $a_{i,j} = Pr\{q_j @ t+1 | q_i @ t\}$: State Transition Probability Distribution**
- **$B = \{b_j(k)\}$, $b_j(k) = Pr(O_t=v_k | q_t=s_j)$: Observation Symbol Probability Distribution in State j**
- **$\pi = \{\pi_i\}$, $\pi_i = Pr(q_i @ t = 1)$: Initial State Distribution**

D. THE THREE BASIC PROBLEMS FOR HMM'S

Three basic problems of interest must be solved in order to specify the model $\lambda = \{A, B, \pi\}$, and use it in classification applications [3]:

Problem 1: How to compute the probability of the observation sequence $Pr(O|\lambda)$, for $O = \{O_1, O_2, \ldots, O_T\}$.

Problem 2: How to compute the most optimal state sequence $I = \{i_1, i_2, \ldots, i_T\}$, for a given observation sequence $O = \{O_1, O_2, \ldots, O_T\}$ and model λ.

Problem 3: How to adjust the model parameters $\lambda = \{A, B, \pi\}$ to maximize $Pr(O|\lambda)$.

Problem 1 is an evaluation problem, because we want to evaluate the probability $Pr(O|\lambda)$, for the specific model. The hidden part of the model is the state sequence I that we attempt to find in Problem 2. Note that there are many possible state sequences, but only one is optimal. Finally in Problem 3, we adjust the parameters A, B, and π of the model λ, so that the probability $Pr(O|\lambda)$ is maximum, i.e., we attempt to optimize the model λ.

E. SOLUTIONS TO THE THREE HMM PROBLEMS

1. Problem 1: Evaluation of $Pr(O|\lambda)$

Problem 1 deals with evaluating the probability of the observation sequence O, given the model λ, i.e. $Pr(O|\lambda)$. Using basic probability principles, it is the summation of the conditional probability $Pr(O,I|\lambda)$ over all possible state sequences I [3]:

$$\Pr(O \mid \lambda) = \sum_{all\ I} \Pr(O \mid I, \lambda) \Pr(I \mid \lambda), \tag{2.10}$$

This evaluation first requires the definition of $\Pr(O,I|\lambda)$, the joint probability that the observation $O=\{O_1, O_2, \ldots, O_T\}$ and the state sequence $I=\{i_1, i_2, \ldots, i_T\}$ occur at the same time, given the model λ [6]. Using the Bayes' rule it is computed as:

$$\Pr(O, I \mid \lambda) = \Pr(O \mid I, \lambda)\Pr(I \mid \lambda). \tag{2.11}$$

Since we assume independence of the observation, we can write:

$$\Pr(O \mid I, \lambda) = \prod_{t=1}^{T} \Pr(O_t \mid i_t, \lambda) = b_{i_1}(O_1) b_{i_2}(O_2) \ldots b_{i_T}(O_T), \tag{2.12}$$

and

$$\Pr(I \mid \lambda) = \pi_{i_1} a_{i_1 i_2} a_{i_2 i_3} \cdots a_{i_{T-1} i_T}. \tag{2.13}$$

Therefore, replacing eqs (2.10), and (2.13) into (2.11) leads to:

$$\Pr(O \mid \lambda) = \sum_{all\ i} \pi_{i_1} b_{i_1}(O_1) a_{i_1 i_2} b_{i_2}(O_2) a_{i_2 i_3} b_{i_3}(O_3) \cdots a_{i_{T-1} i_T} b_{i_T}(O_T). \tag{2.14}$$

As an example of this computation, Let' s apply the above result to the urn-ball example considered earlier in section C and in Figure 2.3. The components of the model $\lambda(A, B, \pi)$ are:

$$A = \begin{bmatrix} 0.5 & 0.3 & 0.2 \\ 0 & 0.8 & 0.2 \\ 0 & 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0.3 & 0.5 & 0.2 \\ 0.4 & 0.4 & 0.2 \\ 0.6 & 0.3 & 0.1 \end{bmatrix}, \quad \pi = \{0.9, 0.1, 0\}.$$

The observation sequence is:

$$O = \{1, 2, 2, 3\},$$

with the number of states N=3, clock times T=4, and the symbol dimension M=3. Recall that we don't know the state sequence, i.e., we cannot identify which specific urn, the color balls came from. Using eq (2.14) we can find the hidden state sequence I, by picking that which leads to the highest probability $\Pr(O,I|\lambda)$. For example, possible state sequence that we can use are I={1,1,1,1} (which means that all balls come from the first urn), or I={1,1,1,2}. Note that this model is a left-to-right model, as the state sequence doesn't go backwards, thus, for instance the state sequence I={1,3,2,2} is not valid as we can't go from the 3rd to the 2nd state. Therefore:

for I = {1, 1, 1, 2}: (first 3 balls from 1st urn, last one from 2nd urn)

$$\begin{aligned}
&\pi_{i_1} b_{i_1}(O_1) a_{i_1 i_2} b_{i_2}(O_2) a_{i_2 i_3} b_{i_3}(O_3) \cdots a_{i_{T-1} i_T} b_{i_T}(O_T) \\
&= \pi_{i_1} b_{i_1}(1) a_{i_1 i_2} b_{i_2}(2) a_{i_2 i_3} b_{i_3}(2) \cdots a_{i_{T-1} i_T} b_{i_T}(3) \\
&= \pi_1 b_{11} a_{11} b_{12} a_{11} b_{12} a_{12} b_{23} \\
&= 0.9 \cdot 0.3 \cdot 0.5 \cdot 0.5 \cdot 0.5 \cdot 0.5 \cdot 0.3 \cdot 0.2 \\
&= 0.0010125
\end{aligned}$$

$\vdots$

for I = {3, 3, 3, 3}: (all balls from 3rd urn)

$$\begin{aligned}
&\pi_{i_1} b_{i_1}(O_1) a_{i_1 i_2} b_{i_2}(O_2) a_{i_2 i_3} b_{i_3}(O_3) \cdots a_{i_{T-1} i_T} b_{i_T}(O_T) \\
&= \pi_{i_1} b_{i_1}(1) a_{i_1 i_2} b_{i_2}(2) a_{i_2 i_3} b_{i_3}(2) \cdots a_{i_{T-1} i_T} b_{i_T}(3) \\
&= \pi_1 b_{31} a_{33} b_{32} a_{33} b_{32} a_{33} b_{33} \\
&= 0.9 \cdot 0.6 \cdot 1 \cdot 0.3 \cdot 1 \cdot 0.3 \cdot 1 \cdot 0.1 \\
&= 0.00486
\end{aligned}$$

As we can see, the number of combinations of all possible sequences is very large, even though the number of the observations T is relatively small. In this problem (T=4), the direct computation of $\Pr(O|\lambda)$ requires $(2T-1)N^T$ multiplications and N^T-1 additions, which is too expensive for real applications. For instance, the number of

computations is $(2*100-1)3^{100}+3^{100}-1$ when T=100 and N=3. Therefore, the more efficient forward-backward procedure was introduced to solve this problem [3, 6].

a) Forward Procedure:

We define the forward variable, $\alpha_t(i)$, as the probability of the partial observation sequence $\{O_1, O_2, \ldots, O_t\}$, until time t and state q_i at time t, given the model λ, such as:

$$\alpha_t(i) = \Pr(O_1, O_2, \ldots, O_t, i_t = q_i | \lambda). \tag{2.15}$$

The forward algorithm defined below is used to evaluate all possible $\alpha_t(i)$ variables:

Initialization:

$$\alpha_1(i) = \pi_i b_i(O_1), \qquad 1 \le i \le N \tag{2.16}$$

Recursion:

$$\alpha_{t+1}(j) = [\sum_{i=1}^{N} \alpha_t(i) a_{ij}] b_j(O_{t+1}) \qquad for\ t = 1, 2, \ldots, T\text{-}1, \quad 1 \le j \le N \tag{2.17}$$

Termination:

$$\Pr(O | \lambda) = \sum_{i=1}^{N} \alpha_T(i). \tag{2.18}$$

Eq (2.18) is also know as the Baum-Welch probability. As we can see from the recursion, the forward procedure gets its name from the fact that α_{t+1} is evaluated using the previous value of the forward variable α_t. Computing $\Pr(O|\lambda)$ using this procedure requires only N^2T calculations instead of the $2TN^T$-1 required by the direct definition [3]. For example direct computation requires $2TN^T=7.4\cdot10^{16}$ computations, while the forward procedure requires $N^2T=1920$ for N=8 and T=30.

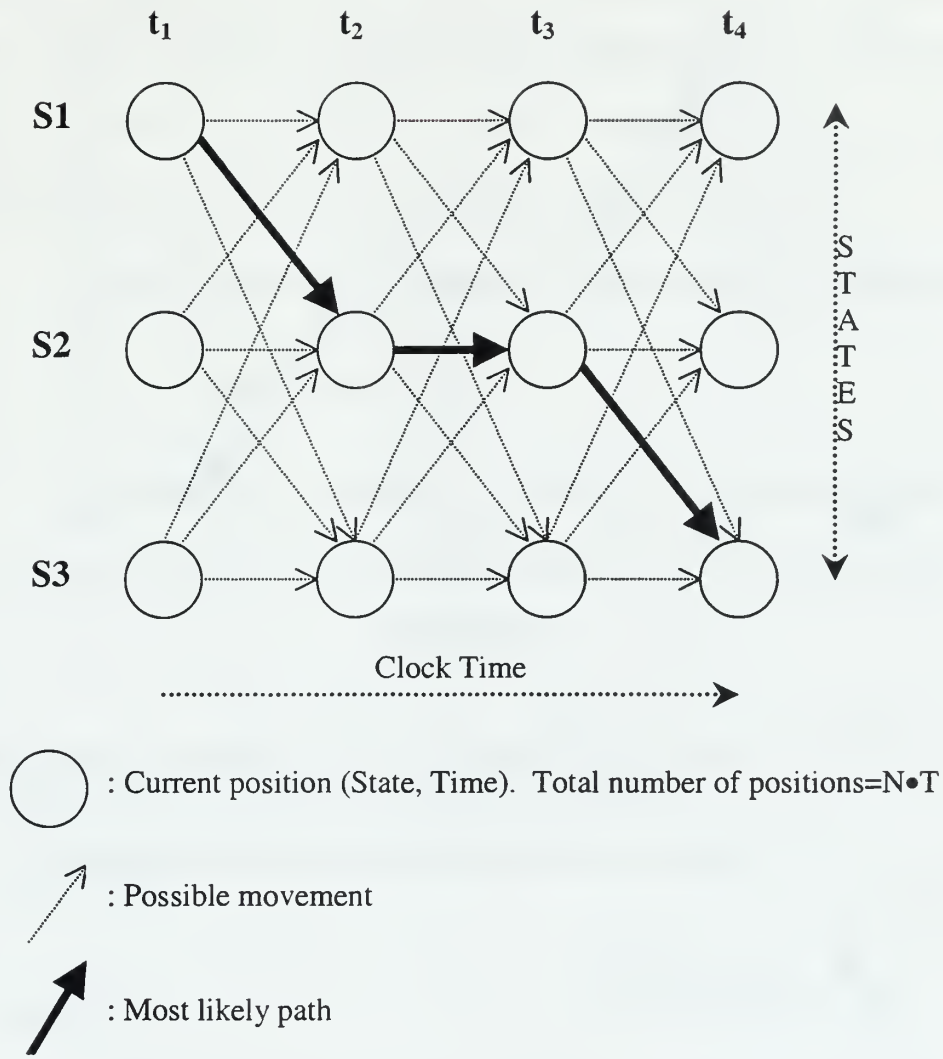


Figure 2.4a Trellis Diagram for T=4, and N=3, ergotic model

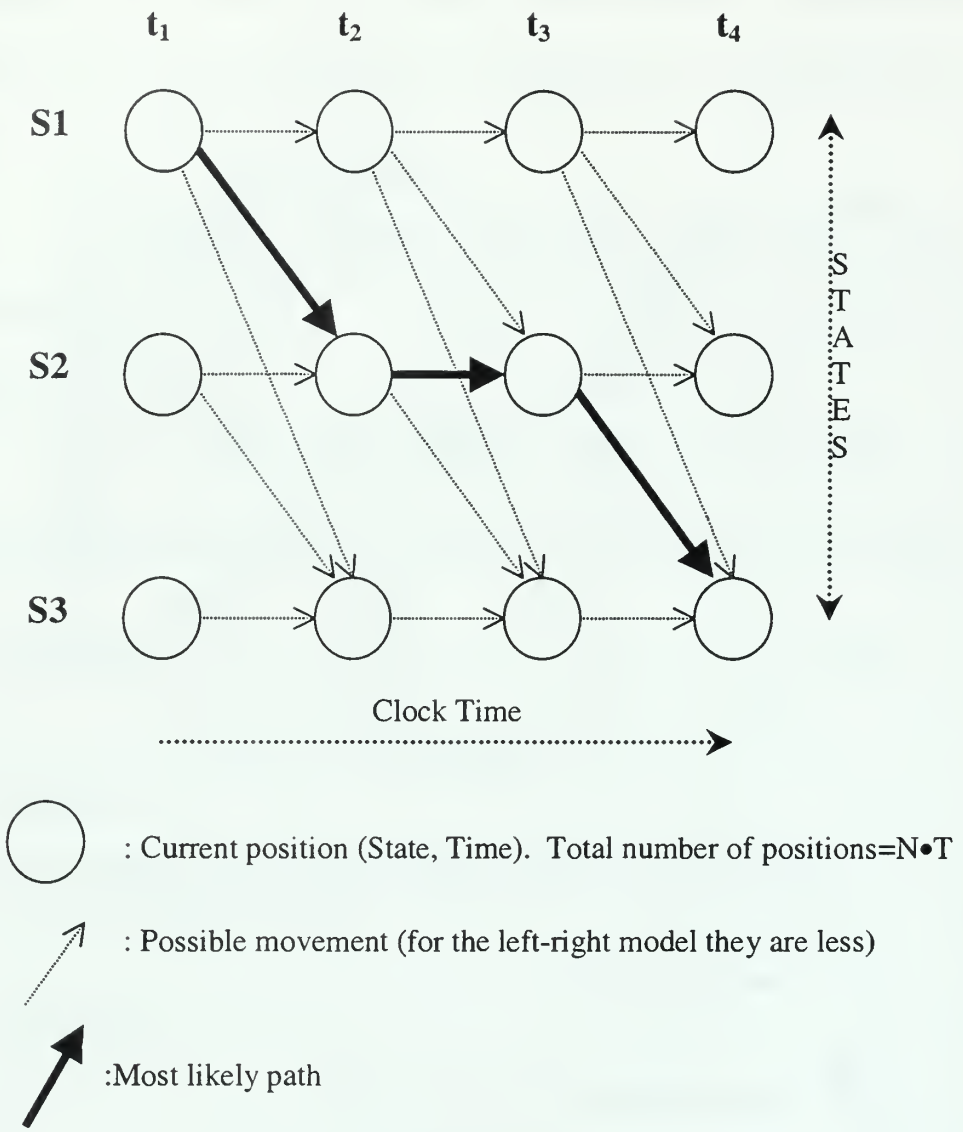


Figure2.4b Trellis Diagram for T=4, and N=3, left-to-right model as used for example in Figure 2.3. Fewer possible paths than in ergotic model are allowed.

The computational savings obtained in the forward procedure are illustrated in Figure 2.4a and 2.4b which present all possible combinations of state

sequences from clock time t_1 to t_4 in a trellis diagram. Figure 2.4a presents the trellis diagram for an ergotic model, and T=4, N=3. Each calculation of $\alpha_t(i)$ with the forward procedure only requires the computation of the values $\alpha_{t-1}(j)$, for $1 \leq j \leq N$. Note that the procedure discards the routes less likely to occur, so that the probabilities through the previously discarded routes do not get reevaluated at the next iteration time.

Finally, the probability $\Pr(O|\lambda)$ is obtained by the summation in eq (2.18).

b) Backward Procedure:

Similarly, we define the backward variable $\beta_t(i)$ as the probability of the partial observation sequence from t+1 to the end, given q_i at time t and the model λ [3], where:

$$\beta_t(i) = \Pr(O_{t+1}, O_{t+2}, \ldots O_T | i_T = q_i, \lambda). \tag{2.19}$$

$\beta_t(i)$ is computed recursively from t=T down to t=1 as follows:

Initialization:

$$\beta_T(i) = 1, \qquad 1 \leq i \leq N \quad . \tag{2.20}$$

Recursion:

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(i), \qquad t = T\text{-}1, T\text{-}2, \ldots, 1 \qquad , 1 \leq i \leq N \,. \tag{2.21}$$

Note that the backward variable is issued to re-estimate the model λ (Problem 3), and that the number of computations is decreased to N^2T.

Therefore, using the definitions of the forward and backward variables, and according to [6]:

$$\Pr(O|\lambda) = \sum_{i=1}^{N} \alpha_t(i) \beta_t(i), \quad t = 1, \cdots, T. \tag{2.22}$$

Urn-Ball Example:

Forward and backward variables are evaluated for the urn-ball example considered earlier. Recall for this example:

$$A = \begin{bmatrix} 0.5 & 0.3 & 0.2 \\ 0 & 0.8 & 0.2 \\ 0 & 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0.3 & 0.5 & 0.2 \\ 0.4 & 0.4 & 0.2 \\ 0.6 & 0.3 & 0.1 \end{bmatrix}, \quad \pi = \{0.9,\ 0.1,\ 0\},$$

$$O = \{1,\ 2,\ 2,\ 3\}.$$

Thus, using eq (2.16) leads to:

$$a_1(1) = \pi_1 b_1(O_1) = 0.9 \cdot b_1(1) = 0.9 \cdot 0.3 = 0.27$$

$$a_1(2) = \pi_2 b_2(O_1) = 0.1 \cdot b_2(1) = 0.1 \cdot 0.4 = 0.04$$

$$a_1(3) = \pi_3 b_3(O_1) = 0 \cdot b_3(1) = 0.$$

Next, using the recursion formula (2.17) leads to the following values for the forward variable $\alpha_t(i)$ for t=1, 2, . . . , T-1 and $1 \le i \le N$. For example,

$$\begin{aligned} \alpha_2(1) &= \left(\sum_{i=1}^{N} \alpha_1(i) a_{ij}\right) \cdot b_1(O_2) \\ &= \left(\sum_{i=1}^{N} \alpha_1(i) a_{i1}\right) \cdot b_1(2) \\ &= \left(\alpha_1(1) \cdot a_{11} + \alpha_1(2) \cdot a_{21} + \alpha_1(3) \cdot a_{31}\right) \cdot 0.5 \\ &= \left(0.27 \cdot 0.5 + 0.04 \cdot 0 + 0\right) \cdot 0.5 \\ &= 0.0675 \end{aligned}$$

$$\alpha_2(2) = (\sum_{i=1}^{N} \alpha_1(i) a_{ij}) \cdot b_2(O_2)$$
$$= (\sum_{i=1}^{N} \alpha_1(i) a_{i2}) \cdot b_2(2)$$
$$= (\alpha_1(1) \cdot a_{12} + \alpha_1(2) \cdot a_{22} + \alpha_1(3) \cdot a_{32}) \cdot 0.4$$
$$= (0.27 \cdot 0.3 + 0.04 \cdot 0.8 + 0) \cdot 0.4$$
$$= 0.0452$$

$$\alpha_2(3) = (\sum_{i=1}^{N} \alpha_1(i) a_{ij}) \cdot b_3(O_2)$$
$$= (\sum_{i=1}^{N} \alpha_1(i) a_{i3}) \cdot b_3(2)$$
$$= (\alpha_1(1) \cdot a_{13} + \alpha_1(2) \cdot a_{23} + \alpha_1(3) \cdot a_{33}) \cdot 0.3$$
$$= (0.27 \cdot 0.2 + 0.04 \cdot 0.2 + 0) \cdot 0.3$$
$$= 0.0186.$$

All values of the forward variable are shown below in the matrix A_{fw}:

$$A_{fw} = \begin{bmatrix} 0.27 & 0.04 & 0 \\ 0.06 & 0.0452 & 0.0186 \\ 0.016875 & 0.022564 & 0.012342 \\ 0.0016875 & 0.00462274 & 0.00202298 \end{bmatrix},$$

which is of dimension: 4x3 (TxN). Similarly, we compute the backward variable, according to eqs (2.20) & (2.21) starting from t=T. Recall from eq (2.20) that:

$$\beta_4(i) = 1, \qquad \forall i = 1,2,3.$$

Next, using the recursion formula given in eq (2.21) to compute $\beta_3(i)$ leads to:

$$\begin{aligned}\beta_3(1) &= \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(i) \\ &= \sum_{j=1}^{N} a_{1j} b_j(O_4) \beta_4(1) \\ &= \sum_{j=1}^{N} a_{1j} b_j(3) \beta_4(1) \\ &= \left[a_{11} \cdot b_1(3) + a_{12} \cdot b_2(3) + a_{13} \cdot b_3(3)\right] \cdot \beta_4(1) \\ &= (0.5 \cdot 0.2 + 0.3 \cdot 0.2 + 0.2 \cdot 0.1) \cdot 1 \\ &= 0.18,\end{aligned}$$

All values for the backward variable $\beta_t(i)$ are shown below in the matrix B_{bw} as:

$$B_{bw} = \begin{bmatrix} 0.027582 & 0.022152 & 0.009 \\ 0.0726 & 0.0636 & 0.03 \\ 0.18 & 0.18 & 0.1 \\ 1 & 1 & 1 \end{bmatrix}.$$

Note, that the values in the last row are always 1, according to eq (2.20). At this point, we can evaluate $\Pr(O|\lambda)$, using the forward variable, according to eq (2.18), which leads to:

$$\begin{aligned}\Pr(O \mid \lambda) &= \sum_{i=1}^{N} \alpha_T(i) = \sum_{i=1}^{N} \alpha_4(i) = \alpha_4(1) + \alpha_4(2) + \alpha_4(3) + \alpha_4(4) \\ &= 0.0016875 + 0.00462274 + 0.00202298 \\ &= 0.00833322000000.\end{aligned}$$

2. Problem 2: Optimal State Estimation

Problem 2 deals with finding the optimal state sequence I for a given observation sequence O. One possible solution to this problem is to maximize the expected number of correct individual states by choosing the states i_t that are more likely to occur [3]. This computation uses the variable $\gamma_t(i)$, defined as the probability of being in state q_i at time t, given the observation O and the model λ [3]:

$$\gamma_t(i) = \Pr(i_t = q_i \mid O, \lambda). \tag{2.23}$$

$\gamma_t(i)$ can be expressed as:

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\Pr(O \mid \lambda)}, \tag{2.24}$$

where $\alpha_t(t)$, $\beta_t(t)$ are the forward and backward variables, defined earlier. Replacing eq (2.23) into eq (2.14) leads to:

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)}. \tag{2.25}$$

Note that the normalization factor $\Pr(O|\lambda)$ in the denominator of eq (2.24) is needed to make $\gamma_t(i)$ a conditional probability, which leads to the following constrain being satisfied:

$$\sum_{i=1}^{N} \gamma_t(i) = 1. \tag{2.26}$$

Finally, the optimal state sequence i_i can be obtained by:

$$i_t = \arg\max_{1 \le i \le N} [\gamma_t(i)], \qquad 1 \le t \le T. \tag{2.27}$$

A more efficient approach to compute the optimal state sequence uses a decoding based on dynamic programming called Viterbi algorithm and shown in Table 2.4. The Viterbi algorithm is designed to find the best path (sequence) which maximizes the probability $\Pr(O, I|\lambda)$ [3].

Table 2.4 Viterbi Algorithm

Initialization:

$$\delta_1(i) = \pi_i b_i(O_1), \quad 1 \le i \le N$$
$$\varphi_1(i) = 0$$

Recursion:

$$\delta_t(j) = \max_{1 \le i \le N} [\delta_{t-1}(i) a_{ij}] b_j(O_t), \quad for\, 2 \le t \le T,\ 1 \le j \le N$$
$$\varphi_t(j) = \arg\max_{1 \le i \le N} [\delta_{t-1}(i) a_{ij}]$$

Termination:

$$P^* = \max_{1 \le i \le N} [\delta_T(i)]$$
$$i_T^* = \arg\max_{1 \le i \le N} [\delta_T(i)]$$

Path (State Sequence) backtracking:

$$i_t^* = \varphi_{t+1}(i_{t+1}^*), \quad For\ t = T-1, T-2, \cdots, 1$$

Application:

As an application, let's go back to example 3 defined earlier in section 3 to find the optimal sequence i_t, given the model λ and the observation O. Recall that, for the given model:

$$A = \{a_{ij}\} = \begin{bmatrix} 0.5 & 0.3 & 0.2 \\ 0 & 0.8 & 0.2 \\ 0 & 0 & 1 \end{bmatrix}, \quad B = \{b_{jk}\} = \begin{bmatrix} 0.3 & 0.5 & 0.2 \\ 0.4 & 0.4 & 0.2 \\ 0.6 & 0.3 & 0.1 \end{bmatrix}, \quad \pi = \{0.9, 0.1, 0\}, \quad O = \{1, 2, 2, 3\}$$

Thus, according to Table 2.4, we get:

$$\delta_1(i) = \pi_i b_i(O_1), \quad 1 \le i \le N$$
$$\delta_1(1) = \pi_1 b_1(O_1) = 0.9 \cdot 0.3 = 0.27$$
$$\delta_1(2) = \pi_2 b_2(O_1) = 0.1 \cdot 0.4 = 0.04$$
$$\delta_1(3) = \pi_3 b_3(O_1) = 0$$
$$\varphi_1(i) = 0$$
$$\delta_t(j) = \max_{1 \le i \le N}[\delta_{t-1}(i)a_{ij}]b_j(O_t), \; 1 \le j \le N, so:$$
$$\delta_2(j) = \max_{1 \le i \le N}[\delta_1(i)a_{ij}]b_j(O_2)$$
$$\delta_2(1) = \max_{1 \le i \le N}[\delta_1(i)a_{i1}]b_1(O_2) = \max[\delta_1(1)a_{11}, \delta_1(2)a_{21}, \delta_1(3)a_{31}]b_1$$

Similarly, after computing all δ values, we define the matrix Δ, such as:

$$\Delta = \{\delta_t(j)\} = \begin{bmatrix} 0.27 & 0.04 & 0 \\ 0.0675 & 0.0324 & 0.0162 \\ 0.016875 & 0.010368 & 0.00486 \\ 0.0016875 & 0.00165888 & 0.000486 \end{bmatrix}$$

Then, we compute the φ variable, defined in Table 2.3 as:

$$\varphi_t(j) = \arg\max_{1 \le i \le N}[\delta_{t-1}(i)a_{ij}].$$

For example, $\varphi_2(1)$ is defined as:

$$\varphi_2(1) = \arg\max_{1 \le i \le N}[\delta_1(i)a_{i1}] = \arg\max[\delta_1(1)a_{11}, \delta_1(2)a_{21}, \delta_1(3)a_{31}]$$
$$= \arg\max[0.27 \cdot 0.5, 0.04 \cdot 0, 0] = 1.$$

Similarly, we compute all other $\varphi_t(j)$ variable, and define the Φ matrix:

$$\Phi = \{\varphi_t(j)\} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix}.$$

Next, we find the last state at time T=4 defined as:

$$i_T^* = \arg\max_{1 \le i \le N}[\delta_T(i)],$$

which leads to:

$$i_4^* = \arg\max_{1\le i\le N}[\delta_4(i)] = \arg\max[\delta_3(1),\delta_3(2),\delta_3(3)]$$
$$= \arg\max[\delta_3(1),\delta_3(2),\delta_3(3)] = \arg\max[0.0016875 \quad 0.0016588 \quad 0.000486]$$
$$= 1.$$

Finally, the other state sequence are defined using the backtracking method of the Viterbi algorithm given in Table 2.4, which leads to:

$$i_t^* = \varphi_{t+1}(i_{t+1}^*), \quad for\ t = 3,2,1.$$
$$i_3^* = \varphi_4(i_4^*) = \varphi_4(1) = 1$$
$$i_2^* = \varphi_3(i_3^*) = \varphi_3(1) = 1$$
$$i_1^* = \varphi_2(i_{2.}^*) = \varphi_2(1) = 1.$$

Therefore, the optimal state sequence for this model and observation sequence is given by $i_t=\{1,1,1,1\}$, which shows that all balls come from the first urn. Note that repeating the same experiment with the observation sequence, $O=\{1,2,3,3\}$, results in $i_t=\{1,2,2,2\}$. Further, note that we always expect a forward moving state sequence, since our model is a left-right model. In addition, we can also use the variable $\delta_t(j)$ to evaluate $Pr(O|\lambda)$, and thus we introduce the Viterbi probability Pr_v, such as:

$$Pr_v = \max_{1\le i\le N}\{\delta_T(i)\}. \tag{2.28}$$

Using eq (2.28) leads to:

$$Pr_v = \max_{1\le i\le 3}\{\delta_4(i)\} = \max\{\delta_4(1),\delta_4(2),\delta_4(3)\}$$
$$= \max\{0.0016875 \quad 0.00165888 \quad 0.000486\}$$
$$= 0.0016875,$$

which is consistent (meaning in the same range) to the value 0.00833 found earlier using the Baum-Welch probability eq (2.18). Note that we can't expect to find the same exact value (since theoretically is not the same), but we can use both probabilities to reconfirm the decision.

3. Problem 3: Re-estimation of Model Parameters

The last problem deals with adjusting the model parameters (A, B, π), so that probability $Pr(O|\lambda)$ is maximum. To that end, we define the variable $\xi_t(i,j)$ as the probability of a path being in state q_i at time t and making a transition to state q_i at time t+1, given the observation sequence and the model, such as [3]:

$$\xi_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\Pr(O \mid \lambda)}. \tag{2.29}$$

Recalling the definition of $\gamma_t(i)$ given earlier in eq (2.25) as the the probability of being in state q_i at time t, given the observation O and the model λ, and using eqs (2.23) and (2.24), we can relate $\gamma_t(i)$ to $\xi_t(i)$, by summing $\xi_t(i)$ over all states j, which leads to:

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j). \tag{2.30}$$

Similarly, summing $\gamma_t(i)$ and $\xi_t(i)$ for all t's, leads to the following result[3]:

$$\sum_{t=1}^{T-1} \gamma_t(i) = \textit{Expected number of transitions made from state } q_i, \tag{2.31}$$

and:

$$\sum_{t=1}^{T-1} \xi_t(i,j) = \textit{Expected number of transitions made from state } q_i \textit{ to state } q_j. \tag{2.32}$$

Finally, using eqs (2.31), (2.32) and the definitions of the model parameters, we can reestimate the model, according to the Baum-Welch formulas:

$$1.\ \overline{\pi_i} = \gamma_1(i), \quad 1 \le i \le N, \tag{2.33}$$

$$2.\ \overline{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}, \tag{2.34}$$

$$3.\ \overline{b}_j(k) = \frac{\sum_{\substack{t=1 \\ O_t=k}}^{T} \gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)} \quad . \tag{2.35}$$

We then continue reestimating our model (applying the new model to the variables γ and ξ), with the re-estimations formulas defined above, until we reach convergence, i.e., so that:

$$\lambda_{new} \approx \lambda_{old}\,.$$

Application: Urn-ball example.

Next, we apply the re-estimation formulas to our urn-ball model described earlier:

Recall:

$$A = \{a_{ij}\} = \begin{bmatrix} 0.5 & 0.3 & 0.2 \\ 0 & 0.8 & 0.2 \\ 0 & 0 & 1 \end{bmatrix},\ B = \{b_{jk}\} = \begin{bmatrix} 0.3 & 0.5 & 0.2 \\ 0.4 & 0.4 & 0.2 \\ 0.6 & 0.3 & 0.1 \end{bmatrix},\ \pi = \{\pi_i\} = \{0.9,\ 0.1,\ 0\}.$$

For the observation: $O = \{1, 2, 2, 3\}$, and a single iteration we get:

$$A = \{a_{ij}\} = \begin{bmatrix} 0.6256 & 0.2945 & 0.0079 \\ 0 & 0.8984 & 0.1015 \\ 0 & 0 & 1 \end{bmatrix},\ B = \{b_{jk}\} = \begin{bmatrix} 0.4362 & 0.4649 & 0.0988 \\ 0.0712 & 0.5573 & 0.3714 \\ 0 & 0.4697 & 0.5302 \end{bmatrix},$$

$$\Pi = \{\pi_i\} = \{0.8936,\ 0.1063\ 0\}.$$

Note that, 1) the summation all rows of the matrixes A, B and vector π remains equal to 1, and 2) A is still an upper triangular matrix, since our model is let-right. Re-estimating the model for 10 iterations leads to:

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0.3341 & 0.6658 \\ 0 & 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0.3325 & 0.6674 \end{bmatrix},$$

$$\Pi = \{1, 0, 0\}.$$

As we can see from the above results, the matrix A still remains an upper triangular, which means that the model is still a left-right model. Examining the B matrix, we conclude that, if we are in the first state, we will observe only the 1st symbol (1st row, 1st column =1), etc. Finally, the π matrix, shows that procedure starts from the first state.

F. SCALING

Numerical implementations of the forward-backward, Baum-Welch or Viterbi algorithm may lead to underflow problems due to the small numbers involved in the required computations. In addition, the problem may become worse, as the matrix dimensions involved in the computations increase. As a result, scaling is required to avoid such mathematical problems [4]. The basic idea behind the scaling procedure is to multiply the forward and backward variables $\alpha_t(i)$ and $\beta_t(i)$ by a coefficient so that the scaled $\hat{\alpha}_t(i)$ and $\hat{\beta}_t(i)$ are kept within the dynamic range of the computer.

Note that we can rewrite the reestimation formula eq (2.34), in terms of the the forward and backward variables, as [4]:

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\sum_{t=1}^{T} \sum_{j=1}^{N} \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}. \tag{2.36}$$

Consider the forward algorithm used to compute of the forward variable $\alpha_t(i)$, discussed earlier. $\alpha_t(i)$ is multiplied by the "scaling" factor c_t defined as:

$$c_t = \frac{1}{\sum_{i=1}^{N} \alpha_t(i)}, \tag{2.37}$$

leading to the scaled forward variable defined as:

$$\hat{\alpha}_t(i) = c_t \alpha_t(j). \tag{2.38}$$

The scaled coefficient $\hat{\alpha}_t(i)$ can be shown to be equal to [4, pg. 272] :

$$\hat{\alpha}_t(i) = \frac{\sum_{j=1}^{N} \hat{\alpha}_{t-1}(j) a_{ij} b_j(O_t)}{\sum_{i=1}^{N}\sum_{j=1}^{N} \hat{\alpha}_{t-1}(j) a_{ij} b_j(O_t)}. \tag{2.39}$$

We also define a modified forward variable, as:

$$\hat{\hat{\alpha}}_t(i) = \sum_{j=1}^{N} \hat{\alpha}_{t-1}(j) a_{ij} b_j(O_t). \tag{2.40}$$

By induction $\hat{\alpha}_{t-1}(j)$ can be written in terms of $\hat{\hat{\alpha}}_{t-1}(j)$ as:

$$\hat{\alpha}_{t-1}(j) = \left(\prod_{\tau=1}^{t-1} c_\tau \right) \hat{\hat{\alpha}}_{t-1}(j). \tag{2.41}$$

Thus:

$$\hat{\alpha}_t(i) = \frac{\sum_{j=1}^{N} \hat{\hat{\alpha}}_{t-1}(j) \left(\prod_{\tau=1}^{t-1} c_\tau \right) a_{ij} b_j(O_t)}{\sum_{i=1}^{N}\sum_{j=1}^{N} \hat{\hat{\alpha}}_{t-1}(j) \left(\prod_{\tau=1}^{t-1} c_\tau \right) a_{ij} b_j(O_t)} = \frac{\hat{\hat{\alpha}}_t(i)}{\sum_{i=1}^{N} \alpha_t(i)} \quad . \tag{2.42}$$

The scaled backward variable $\hat{\beta}_t(i)$ is defined as:

$$\hat{\beta}_t(i) = c_t \beta_t(i). \tag{2.43}$$

At this point, we can rewrite the reestimation formula given in eq (2.36), using the scaled forward variable such as:

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \hat{\alpha}_t(i) a_{ij} b_j(O_{t+1}) \hat{\beta}_{t+1}(j)}{\sum_{t=1}^{T} \sum_{j=1}^{N} \hat{\alpha}_t(i) a_{ij} b_j(O_{t+1}) \hat{\beta}_{t+1}(j)} \quad . \tag{2.44}$$

Similarly the re-estimation formula for $b_j(k)$ eq (2.35) becomes:

$$\bar{b}_j(k) = \frac{\sum_{\substack{t=1 \\ O_t=k}}^{T-1} \hat{\alpha}_t(j) \hat{\beta}_t(j)}{\sum_{t=1}^{T-1} \hat{\alpha}_t(j) \hat{\beta}_t(j)} . \tag{2.45}$$

Finally, it can be proved that the probability $\Pr(O|\lambda)$ and the Viterbi probability $\Pr_v$ can be computed using the scaling factor [4],which leads to:

$$\Pr(O \mid \lambda) = \frac{1}{\prod_{t=1}^{T} c_t}, \tag{2.46}$$

or,

$$\log[\Pr(O \mid \lambda)] = -\sum_{t=1}^{T} \log c_t \tag{2.47}$$

$$\log(\Pr_v) = \max_{1 \le i \le N} [\phi_T(i)]. \tag{2.48}$$

G. MULTIPLE OBSERVATION SEQUENCES

In real word applications we need to train the HMM using multiple trials, to ensure robustness in the recognition/classification process. Each training trial signal produces an observation sequence. Assume that there are K trials, i.e., K observation sequences. We denote the set of observation sequences $\tilde{O}$ such as:

$$\tilde{O} = \left\{O^{(1)}, O^{(2)}, \cdots, O^{(k)}\right\}, \tag{2.49}$$

where:

$$O^{(k)} = \left\{O_1^{(k)}, O_2^{(k)}, \cdots, O_{T_k}^{(k)}\right\}, \tag{2.50}$$

is the k^{th} observation sequence. Provided that each observation sequence is independent of each other and identically distributed, we want to find the model which maximizes the probability:

$$\Pr(O \mid \lambda) = \prod_{k=1}^{K} \Pr\left(O^{(k)} \mid \lambda\right) = \prod_{k=1}^{K} P_k \ . \tag{2.51}$$

Therefore, the multiple observation reestimation formulas using the scaled variables are:

$$\tilde{a}_{ij} = \frac{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k - 1} \hat{\alpha}_t^k(i) a_{ij} b_j\left(O_{t+1}^{(k)}\right) \hat{\beta}_{t+1}^k(j)}{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k - 1} \hat{\alpha}_t^k(i) \hat{\beta}_t^k(i)}, \tag{2.52}$$

$$\tilde{b}_{ij} = \frac{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k - 1} \hat{\alpha}_t^k(i) \hat{\beta}_t^k(j)}{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k - 1} \hat{\alpha}_t^k(i) \hat{\beta}_t^k(i)}. \tag{2.53}$$

Note that we don't reestimate π_i, so we keep it unchanged through the iterations.

III. ISOLATED WORD RECOGNITION

This section presents the application of HMMs to speech recognition. Specifically, we show how the model parameters have to be selected and adjusted using a limited 4-word recognition example. Our goal is to show how a more generic classifier can be set-up through the speech recognition example. In addition, we point out the potential difficulties one may encounter while setting up and implementing the software for such an application, due to computer numerical precision limitations.

A. GENERAL HMM TRAINING AND TEST PROCEDURE

This section describes the application of HMMs to classification in the context of speech recognition. The overall procedure is illustrated in Figure 3.1. First, labeled data is used to train the model. These specific training signals may be multiple trials of the same type of signal, i.e., belong to the same class, or belong to different classes. For example, multiple trials of the same word may belong to the same signal class in speech recognition applications. In such a case, all words used in the recognition set-up constitute the dictionary. Next, information uniquely characterizing each class needs to be extracted from the signal classes. Thus, each signal is split into T segments, and some useful features extracted from each. Feature vectors may include LCP or cepstral coefficients, energy, etc... Thus, the initial signals are converted into a set of continued-valued vectors. Next, this set gets converted into a sequence of discrete vectors using vector quantization (VQ) [4,5], which will be described further in Section 2. The set of M discrete symbols forms the codebook. For example, assume we have a speech signal divided into 4 segments (i.e., T=4), and that the set of symbols representing one signal segment is a number ranging from 1 to 8. Thus, a possible observation sequence of the

k^{th} signal may be given as $O^k = \{7, 3, 4, 8\}$. At this point, we can check whether the features extraction method and dimension of the codebook M makes sense by comparing the observation sequences obtained for the signals of the same class, as it is reasonable to expect some similarity between the resulting sequences.

Recall that the set of observations derived from each class of training signals is the only information used to train a class-specific model $\lambda\{A, B, \pi\}$. First, an initial estimate of the model is required to apply the Baum-Welch algorithm, as described earlier in Section II. Initial values may be set randomly so that they satisfy the model constraints and converge to the correct type of model, i.e., the initial matrix A is to be selected as upper triangular for left-to-right models so that it converges to an upper triangular matrix. The Baum-Welch algorithm iteratively estimates the model and stops when there is no significant model parameter changes between two successive iterations.

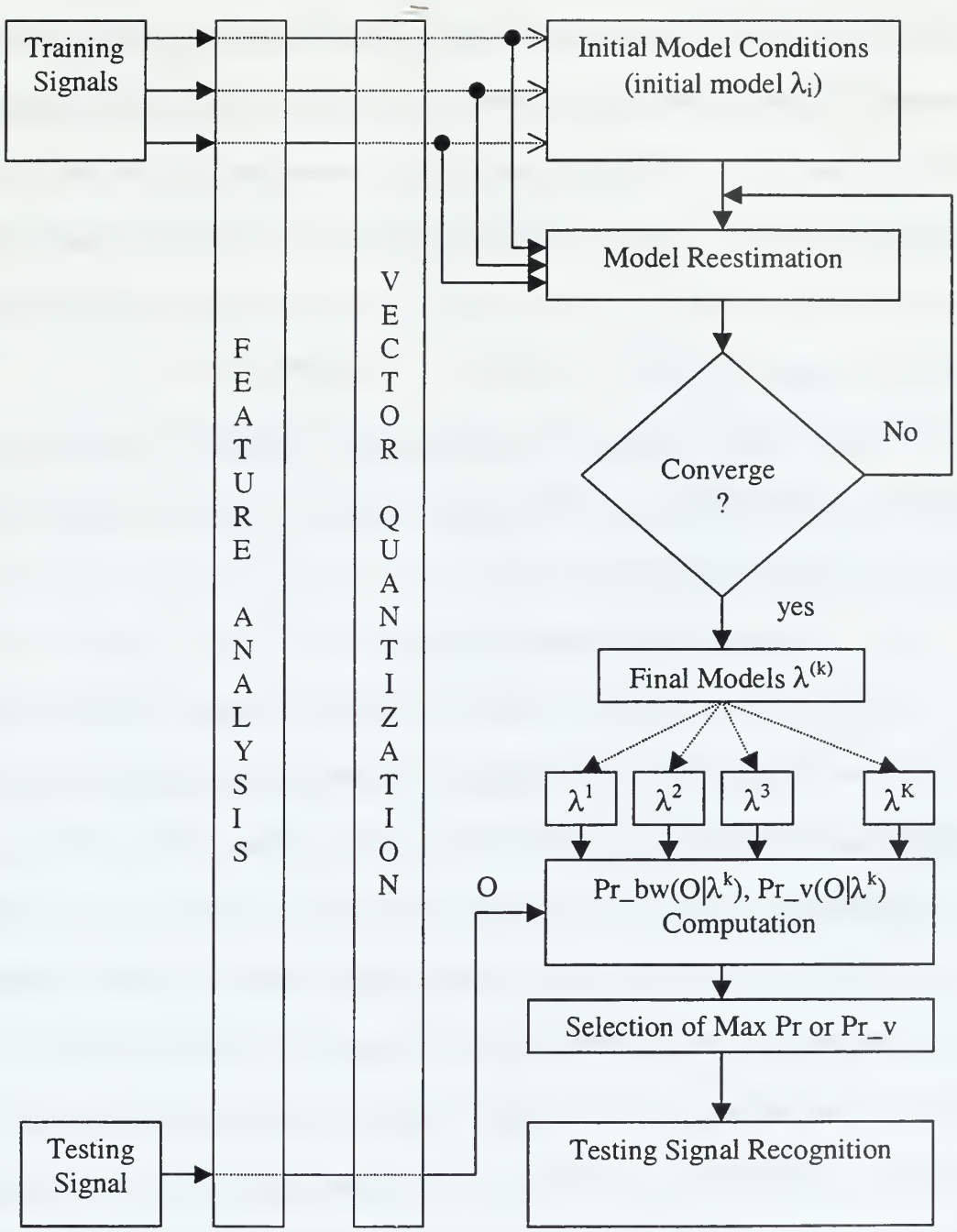


Figure 3.1 HMM General Training and Testing Procedure

Unlabeled signals are used during the testing phase to identify which class they belong to. First, testing data are pre-processed to generate the codebook vectors and corresponding observation sequences, following the same process as that used during the training phase. Next, the set of the probabilities observing the tested signal O, given the k^{th} model, $Pr(O|\lambda^{(k)})$ is computed using either the Baum-Welsh or the Viterbi algorithm for all k^{th} models, as discussed earlier in Section II. Finally, the model type is selected by choosing that with the highest probability, $Pr(O|\lambda^{opt})$.

Next, we describe a simple 4-word recognizer designed to recognize the words: Statistics, Microsoft, Instructor, and Professor. Three trial words are used for training and one word is use for testing for each class.

1. Data Creation and Preparation

All data were recorded on the same machine running Windows-98 Sound Recorder with a sampling rate of 8000Hz sampling and 8bit mono encoding. One male speaker was used. However, note that there are some variations between the trials so that no word is pronounced twice exactly the same way. An energy detector was applied to remove silence before and after each word, resulting in word of about 9000 points. Next, each signal was interpolated to 10000 points to obtain trials with the same length. Finally, each data was sent through a pre-emphasis filter with transfer function $H(z)=1-\alpha z^{-1}$ with $\alpha=0.98$, to emphasize the relative energy of the high-frequency spectrum which contain useful information. The MATLAB software implementation is presented Appendix A.3.

2. Feature Extraction

We varied the length and number of segments (time frame), and the specific type of feature parameters until we obtained a combination which lead to correct classification.

The final set of parameters was derived by each word into T=7 segments using a rectangular window, with an overlap of 10%, where the time length of each trial was about 1sec. The following eight parameters were extracted for later use in VQ from each segment: LPC coefficients from a 7^{th} order filter derived with the covariance method [1], and the energy of the section. Finally, we normalize the value of the energy dividing all values by the max energy.

3. Vector Quantization

Vector quantization (VQ) is a scheme, which maps a sequence of continuous-valued vectors into a sequence with a given number of discrete vectors, called the codebook [7]. Therefore VQ can be viewed as some type of encoding scheme, where the encoder γ assigns a channel symbol $\gamma(x)$ from an ensemble of M symbols to each input vector $x=\{x_0, x_1,..., x_{k-1}\}$ [7]. Note that there is no need to define a decoder, as the discrete sets of parameters never get translated back into the original vector.

Basically, VQ partitions the set of coefficients into M disjoint sets. Each set is represented by a single vector $\{v_m\}1\leq m\leq M$, which is a centroid of the vectors in the coefficient set assigned to the m^{th} region [4].

Note that there is a distortion penalty associated with VQ, as all feature vectors are represented using a set of M codebook vectors. The larger the dimension of the codebook, the smaller is the overall distortion between original feature vectors x and codebook vectors x_r. The distortion measure $d(x,x_r)$ resulting from the codebook selection process can be represented using the Euclidean norm as [7]:

$$d(x,x_r)=\|x-x_r\|^2. \tag{3.1}$$

For our purpose two schemes were applied to derive the codebook vectors: 1) a competitive Neural Networks implementation, and 2) a K-means (or LGB algorithm) algorithm [7, 8].

a) Competitive Neural Network Implementation

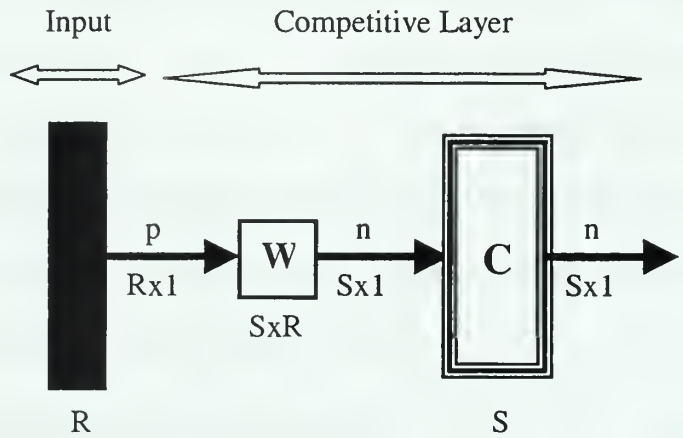


Figure 3.2 Competitive Neural Network

A competitive unsupervised neural network (NN) implementation was selected to compute the codebook, as shown in Figure 3.2 [2]. Basically, this NN can be viewed as a clustering scheme, where weights associated to each neuron are used to assign a symbol M to each word segment. The software implementation is presented in Appendix A.1.

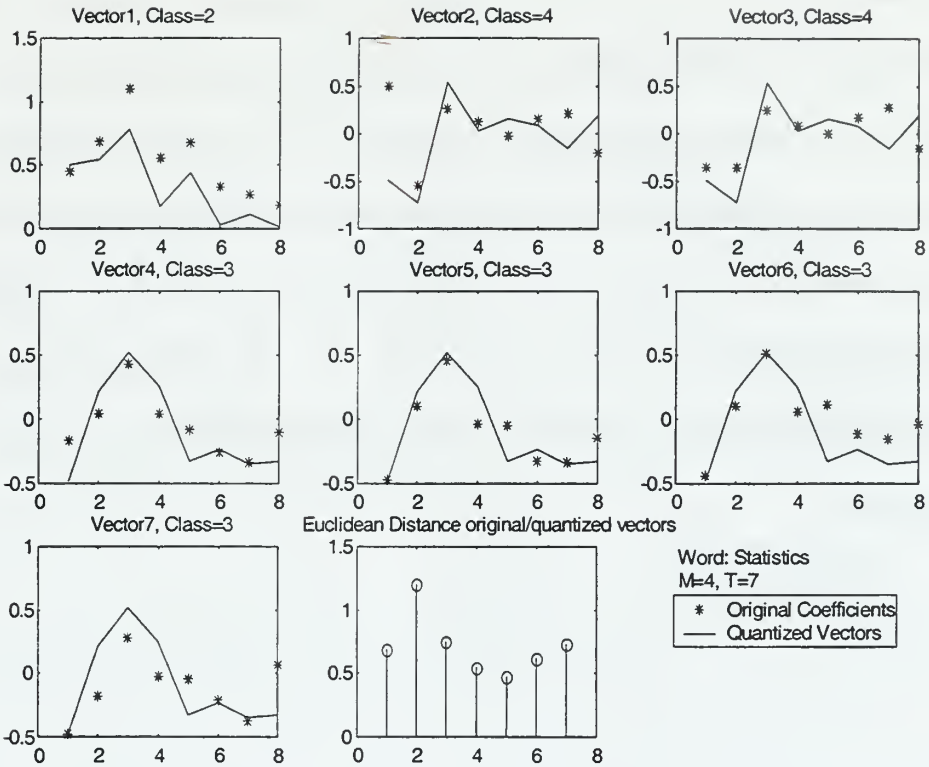


Figure 3.3 Diagram of coefficient vectors and VQ vectors, Euclidean distance for the word "statistics", using a competitive neural network

Figure 3.3 shows seven original feature vectors, and the corresponding quantized vectors obtained with the competitive NN. Training took about 2000 epochs and 50mn with a Pentium-III 450MHz. Each initial feature vector is mapped into one of the M=4 quantized vectors. For example, vectors 2 and 3 get mapped to the same fourth class due to their consistency. Similarly, vectors 5, 6, and 7 get mapped to the 3rd class of the quantized vectors.

b) K-means Scheme

The second method considered is the K-means algorithm, also called the Lloyd or Linde-Buzo-Gray (LBG) algorithm [9]. The software implementation is given

in Appendices A.13 and A.14 [8]. Basically, the K-means algorithm is a clustering scheme, which iteratively finds a set of k, quantized vectors into which all training vectors get mapped to with minimum distortion. The number of clusters increases iteratively by splitting the existing quantized vectors obtained at each iteration, until a desired number of quantized vectors or distortion levels is obtained [9]. Thus, the size of the codebook is a power of two, i.e., 2, 4, 8, 16, etc... The LGB algorithm is illustrated for the word "Microsoft" in Figures 3.3 and 3.4 for codebook sizes equal to 4 and 32 respectively.

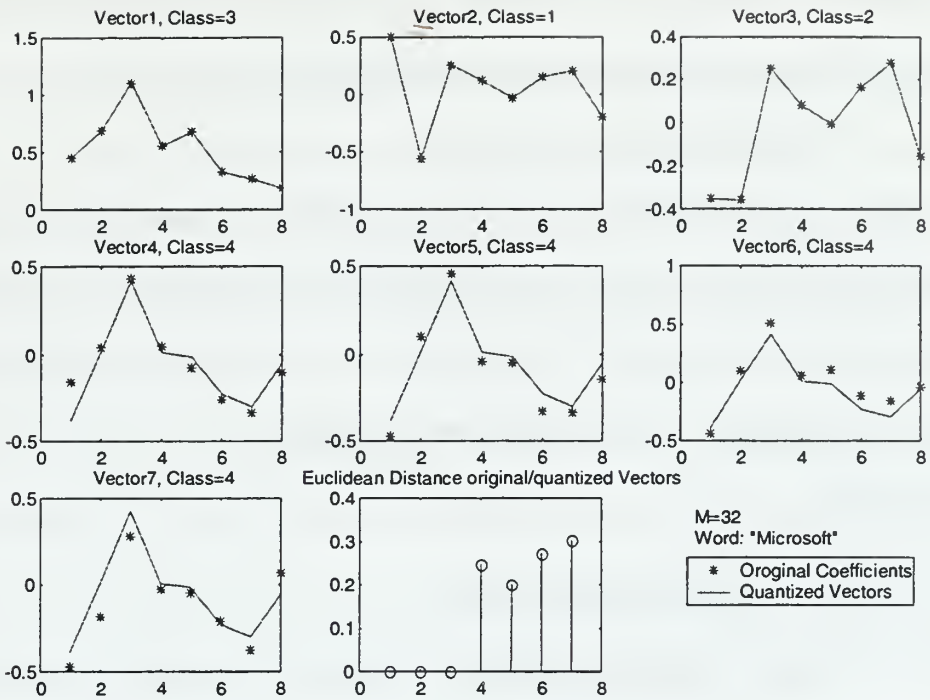


Figure 3.4 Diagram of coefficient vectors and VQ vectors, Euclidean Distance for word "Microsoft," using the K-means (LGB) algorithm and M=32.

Note the distortions are much smaller when M=32 than when M=4. This is to be expected as a larger codebook size allows more flexibility. However, a large codebook may not be always desirable as it may lead problems in evaluating and comparing the resulting probabilities $Pr(O|\lambda)$, as we will see later. The K-means implementation is a little faster than the NN technique, however it is restricted to codebook sizes which are powers of two. No such restriction is needed for the NN implementation.

Note that an initial selection of the number of segments T and the size of the codebook M may be made by observing a small selection of the resulting quantized

vectors. The basic idea is to select a combination of parameters, which lead to some type of consistency in the resulting quantized vectors in a given class.

4. HMM Training

This section describes the application of the HMM training theory presented in Section II to the speech recognition example. The HMM implementation uses MATLAB 5.3, and is presented in Appendix A. MATLAB may not be the most desirable software language as it is relatively slow, however it was selected for ease of implementation to test the concepts.

a) HMM Initial Conditions

Usually, a left-to-right model is preferred over the more general ergodic one in speech applications because model states can be associated with time in a straightforward manner [4]. Thus, we selected a left-to-right model, where the matrix A is upper triangular. In addition, we also prefer the model to start from an early state so that it can pass from all possible states. As a result, we didn't use random values for the vector π, but instead we initialized the first coordinate to be quite large, the second one smaller, and so on, and applied the constrain that the summation of all values in π is equal to one. We use random values satisfying the appropriate constraint for the matrix B.

b) Number of States N

HMM states are called "hidden" because all information about the state sequence is not accessible directly, since the only information available is given by the observations. According to Rabiner [4], there are two schools of thought for the physical meaning of the number of states; the first one states that it represents the number of

sounds (i.e., phonemes) for each word, which is usually a number between 2 and 10. The second interpretation is that it represents the average number of observations in a spoken version of the word. Basically, we can assume that the number of states represents the number of distinct sounds (e.g., phonemes or syllables) of the word. In our case a number of states N equal to 4 was deemed appropriate, since our vocabulary contains only 4 words.

Further investigations may be required when applying HMMs to other types of signals if we cannot relate the number of states to some physical meaning behind the data. At worse, the number of states can be selected by trial and error at that leading to the highest recognition results. Note that selecting too small a number of states may prevent from differentiating between classes. Simulations showed that selecting too large a number of states may result in overly long training time and numerical instability in the implementation which cannot be controlled with scaling.

c) HMM Re-estimation

The HMM re-estimation iterative scheme implemented uses the multiple observation sequence technique described earlier in Section 2. The MATLAB software implementation is given in Appendix A.7 to A.9. First, we re-estimate the model parameters for every different trial, and then we apply these results in the re-estimation formula in eqs (2.52) and (2.53).

The HMM re-estimation step uses the scaled forward and backward variables to avoid numerical instabilities, and its implementation is given in Appendix A.6. In addition, we ran all MATLAB files using a scaled fixed point format with 15 digits (format long). Since, some computations still resulted in "divided by zero" errors

after applying these corrective measures. We forced the denominator quantities to be equal to the smallest floating point number whenever there were found to be smaller.

We used multiple iterations for each single observation HMM re-estimation step until convergence was reached. We noticed that the model doesn't necessarily converge to the same parameters for the same observation, when repeating the re-estimation procedure with random initial conditions. However, correct decision is still achieved.

d) Scoring

The system is ready to test any word and categorize it as one the four class types after the four models $\lambda^{(k)}$, k={1,2,3,4} derived separately for each word (Microsoft, Statistics, Instructor and Professor) are identified. We repeat the same feature extraction scheme for the testing words, using either the neural network or the codebook derived in the K-means algorithm during the training process. Let's assume that the observation of the testing word is O. During testing, we evaluated the probability $Pr_{bw}(O|\lambda^{(k)})$ for k=1,...4, using either the Baum-Welch or the Viterbi algorithm, and selected the model which gives the highest probability $Pr(O|\lambda^{(k)})$ (Appendix A.10, A.11).

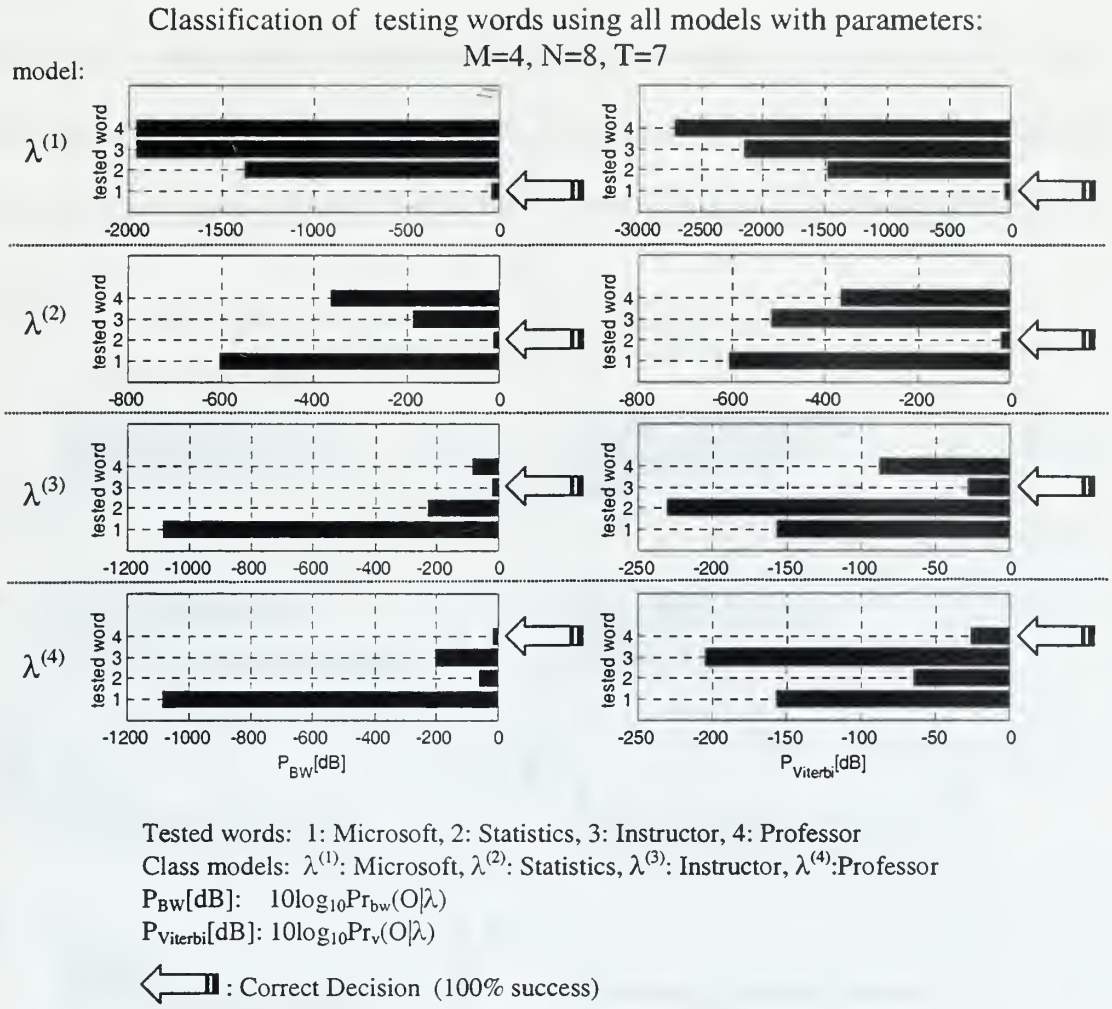


Figure 3.5 Scoring of all 4 testing words (1: Microsoft, 2: Statistics, 3: Instructor, 4: Professor) given each model $\lambda^{(k)}$. Parameters: M=4, N=2, T=7. This system performed 100% successful decisions.

Figure 3.5 presents the results obtained when the number of segments T is equal to 4, the number of symbols M is equal to 4 (computed with the LGB algorithm), and the number of states N is equal to 8. The four horizontal bars contained in each word represent the probabilities $10\log_{10}(P(O|\lambda^{(k)}))$, k=1,...4. Thus, the highest probability is that closer to the 0dB point, which represents the point of probability 1. The left column plots represent the results obtained with the Baum-Welsh algorithm while the right colum plots represent those obtained with the Viterbi algorithm. Recall that the four models

$\lambda^{(1)}$, $\lambda^{(2)}$, $\lambda^{(3)}$, and $\lambda^{(4)}$ were trained with three trials for every class. The software implementation for this testing step is given in Appendix A.11. Results show that correct classification is obtained in all 32 cases, as shown in Figure 3.5.

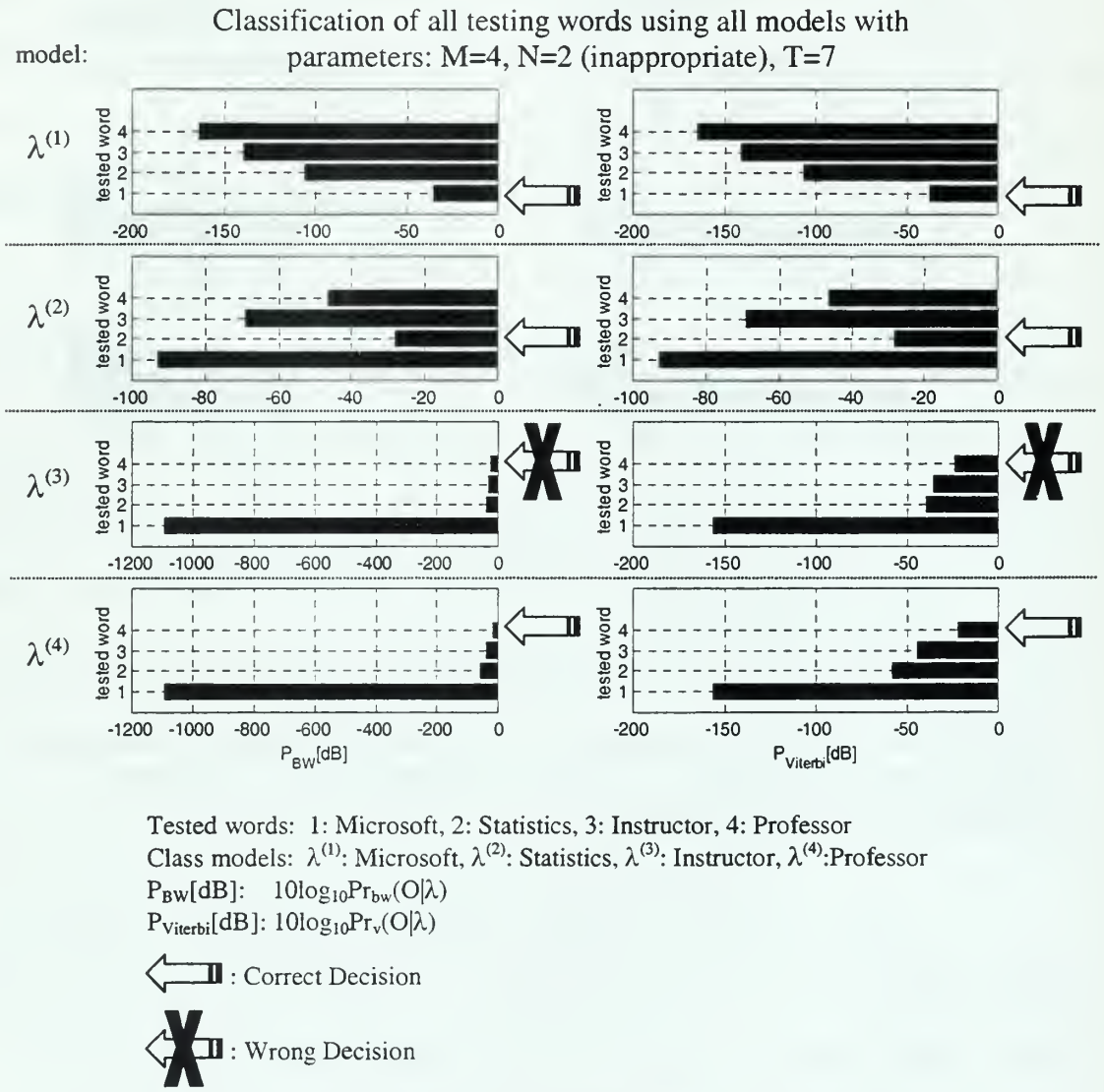


Figure 3.6 Scoring obtained for all four testing words (1: Microsoft, 2: Statistics, 3: Instructor, 4: Professor) given each model $\lambda^{(k)}$. Model Parameters selected: M=4, N=2, T=7. This HMM has too few states (N=2) and cannot classify the word 'instructor' correctly.

Figure 3.6 presents the results obtained when selecting too small a number of states (N=2). All other parameters are kept the same as those in Figure 3.6. Note that the algorithm reaches the wrong decision for the word "instructor" which was found to be "professor."

B. CONCLUSIONS

This section presented an HMM-based classifier applied to a simple 4-word recognition problem. We implemented and tested the classifier using MATLAB, and described how we selected the various parameters which need to be selected to set-up the classifier. Next Chapter considers the application of the HMM-based classifier to seismo-acoustic signals to differentiate between two types of mine-like objects buried in shallow sand.

IV. HMM-BASED CLASSIFICATION OF SEISMO-ACOUSTIC MINE SIGNALS

HMMs can be applied to various types of classification problems. This Section presents the results obtained for the classification of mine-like objects buried in sand. The mine data was obtained from Prof. Muir who heads a buried mine detection project started in November 1996 at the Naval Postgraduate School. Initial results obtained are described in Gagham [10], Fitzpatrick [11], and Hall [12].

The NPS mine project is a continuation of work started at the Applied Research Laboratory of the University of Texas at Austin, and is sponsored by the Office of Naval Research. The main goal of the project is to study the development of a seismo-acoustic sonar for the detection of buried ordnance using guided, seismic interface waves. Earlier ARL and NPS results showed that seismic interface (Rayleigh) waves can be used to detect mine-like objects buried in sand [10-12]. The goal of the current NPS program is to develop an improved seismic source to evaluate the feasibility of using a seismo-acoustic sonar to detect buried ordnance in the beach and surf zones. The seismic waves were generated by two actuators, and were measured by two three-axis sensors-geophones. Buried, mine-like objects, ranging from 71kg to 290kg, and at ranges of up to 5 meters were echo-located by applying a basic polarization filtering signal processing scheme.

This section applies the HMM concepts derived earlier to distinguish between two types of mine-like objects. Two types of experiments were conducted: 1) classification of two different mine-like objects at the same range with multiple weights for each mine

type; and 2) classification of two different mine-like objects at 3 different ranges with the same weight for each mine type.

A. BASIC EXPERIMENT INFORMATION

This section does not present any theoretical, physical or experimental details for this project, as they may be found in in [10-12]. However, we do provide some basic information regarding the physical nature of the signals under study.

The beach site used for collecting the data is a stretch of U.S. Navy-owned beach directly seaward of NPS, Monterey, CA, and shown in Appendix B.2. This area measured roughly 150 feet in length running parallel to the waterline and varied from 20 to 50 feet from the high-to-low water mark. Generally, the sand conditions varied due to the different waterline distance, resulting in changes on some of the sand characteristics, such as density and moisture in multiple depth layers.

The equipment configuration for the mine detection is illustrated in Appendix B.6. Basically, two actuators produce the Rayleigh waves described in [11]. Rayleigh waves have distinctive features that make them identifiable in a complex seismo-acoustic wavefield. The most important property of Rayleigh waves is the elliptical particle motion produced by their passage. Their unique characteristic is the 90° phase shift between their horizontal and vertical components, which results in elliptical motion, as illustrated in Appendix B.7. Thus, the placement and operation of the actuators, as illustrated in Figure 4.1, can be categorized into two different configurations: 1) actuators operated horizontally; and 2) actuators operated vertically, with a 90° relative phase difference between their drive signals [11].

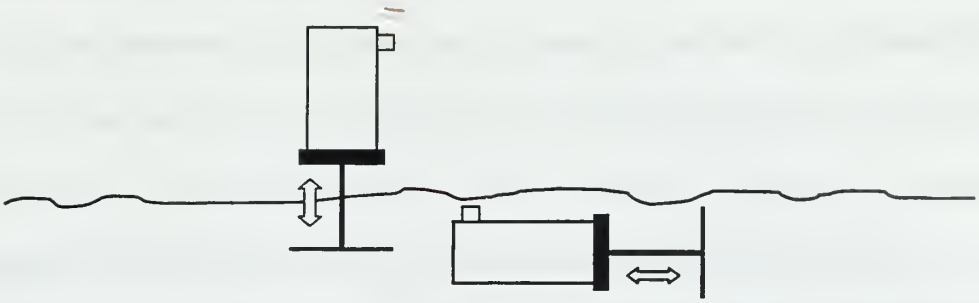


Figure 4.1 Actuator placement for Rayleigh waves generation [11].

The generated seismo-acoustic wave travels through the geophones and meets the target, as illustrated in Appendices B.6 and B.8. Next, the reflected wave passes through the geophones again, and the received signal is used to detect the target presence. The signal is collected, as described in [12], and the signals for the relative x, y, and z-motion are shown in Appendix B.5.

Note that the target reflection is not visible from the raw signals. Thus, vector polarization (VP) [12] was used to extract the Rayleigh waves from the raw information. The VP step takes advantage of the 90° phase shift between vertical and radial components to pull the target information out. This is accomplished by applying the Hilbert Transform to the radial-vertical signal pair [11], the complex crossed-power is computed such as [12]:

$$P_{rv} \equiv r_{hilbert}^{*} \times v_{hilbert} \tag{4.1}$$

the imaginary component of P_{rv} is essentially proportional to the intensity of the seismic wave due to the 90° phase shift between vertical and radial components [12]. The polarity of the imaginary power is associated with the rotation of the elliptical motion of the wave (e.g., a negative value corresponds to an anticlock-wise motion). Real and

imaginary components of the cross-power P_{rv} are illustrated in Appendix B.9. Appendix B.8 shows the incident wave passing the geophone at 10ft. The reflected target signal strength is quite small, and we need to expand the scale to see it, as explained further later.

Two mine-like objects were used: 1) a cylinder weighing 150lb (68kg), 5ft long, 8in (20cm) in diameter, with a ¼-in (0.6cm) wall thickness; and 2) a U.S. Navy power can or "power keg" with the shape of a cylindrical sheet metal can 18in (46cm) high and 24in (61cm) in diameter, weighing 16lb (7kg). These objects are shown in Appendix B.3. Additional weights were used to vary the objects total weight. These additional weights made the maximum weight of the cylinder and the keg equal to 618lb (280kg), and 640lb (290kg) respectively. The cylinder was always buried on its side with the cylindrical axis horizontal and in the direction of the wave propagation. The powder keg was always buried upright, with the cylindrical axis vertical to the direction of the wave propagation. Further details are given in Appendix B.4.

B. SIGNAL SELECTION

Recall that the target signal is visible after processing the raw information using VP. Thus, we use the signals obtained from the imaginary portion of the cross power for our application only. We also focus on the experiments conducted on the 6^{th} and 10^{th} of November 1998, were the cylinder and the keg targets were used, as shown in Appendix B.6. Two parameters were varied during these two days in addition to the sea conditions: target weights and distances from the geophones. Ten trials were conducted for each experiment. The distances between the different pieces of the equipment (actuators, geophones, target) were set as shown in Appendix B.6. As a result, the actuator-

geophone distance was set at 10ft, the actuator-target distance: 16ft, and the actuator-target-geophone (reflected) was 22ft for both targets (cylinder and keg). Appendix B.8 shows the average imaginary cross power obtained for all trials associated with a specific target and weight, scaled near the distance of 22ft. Appendix B.9 shows that the reflected target signal becomes stronger as the target mass increases. Appendices B.10 and B.11 plot the imaginary cross-power signals obtained from the cylinder and the keg targets respectively. We used the signal from the 2^{nd} geophone for the cylinder case, as it is the strongest of the two, and the signal from the 1^{st} geophone for the powder keg, as the 2^{nd} geophone doesn't show any received signal. However, the signal measured at the 2^{nd} geophone for the keg experiment is used as an example of background noise (no target) signal. This background noise is used as part of the testing data, as described later.

C. SIGNAL PREPARATION

Plots contained in Appendix B.10 and B.11 show there is some consistency between the signals obtained from multiple weights of the same target. Thus, we elected to consider all signals associated with a given target to the same class, independent of the weight. Five trials were used for training, and the 6^{th} trial was used for testing. The following weights were used for the keg target: 1) 224lb; 2) 432lb; and 3) 536lb. Similarly, the following weights were used for the cylinder target: 1) 364lb; 2) 468lb 3) 572lb. Next, we assumed that we detected the presence of a non-background signal using some type of threshold detector, and truncated the signals used for the HMM set-up around the position of the target location. Signals used for the HMMs training and classification are shown in Figure 4.2. Note that it would be useless to include the incident signal received at a geophone from the actuators, as it doesn't contain any

information about the target, and is much stronger than the signal reflected from the target. Each signal is 400 data points long. Two classes (keg and cylinder) are generated. At this point, the goal is to recognize the shape of the target, and each class needs to be characterized by a set of features, as explained next.

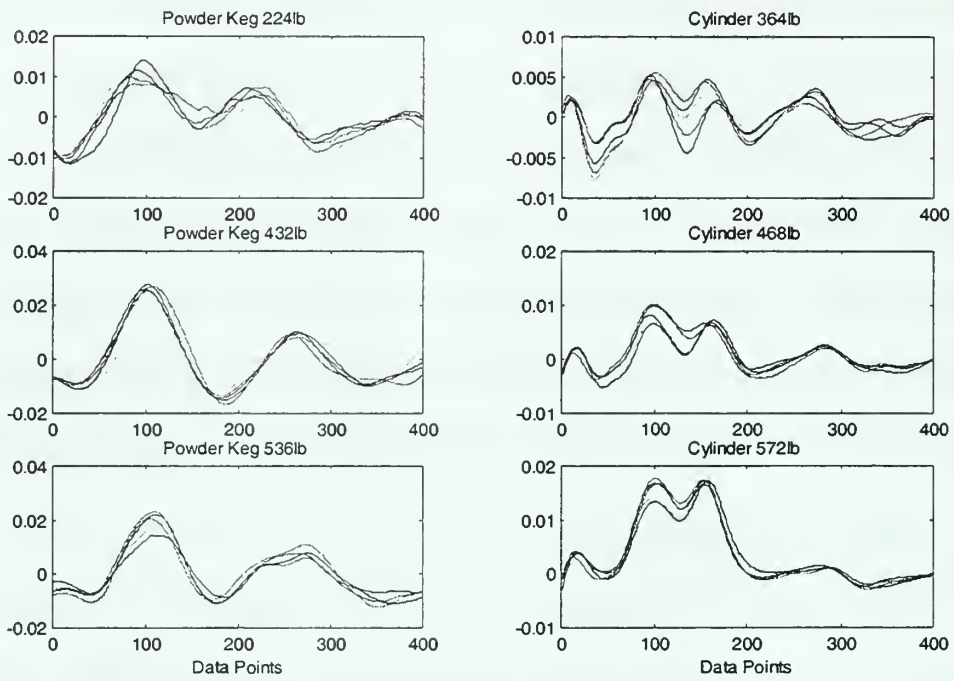


Figure 4.2 Imaginary component cross-power for the keg and cylinder targets for 3 different weights, truncated near the target location. Each plot illustrates 6 different trials (for the same target, and weight)

D. FEATURES EXTRACTION/VECTOR QUANTIZATION

Note in Figure 4.2 that the signals contain little spectral information. The frequency of the Rayleigh waves obtained during the November 6th and 10th experiments was 80Hz. As a result, the signals contain very little useful spectral information. We tried to apply LPC feature extraction, and other similar spectral coefficients, but we found little or no consistency between the trials of a given class. Therefore, we simply used the time-domain information directly. We segmented the signals into two segments

(T=2), and decimated each frame by a factor of 20:1, resulting in 10 data points over each segment, as shown in Appendix C.4. Finally, we normalized those points to compensate for the differences in signal strength coming from different weights, by dividing every value with the maximum one.

We applied the LGB algorithm at the VQ stage and selected M=8 symbols. The code implementation is given in Appendix C.3.

E. HMM TRAINING FOR THE MULTIPLE WEIGHTS EXPERIMENT

HMMs need data to be trained. However, in practice the availability of data may be seriously limited for various reasons and cross-validation needs to be used [13].

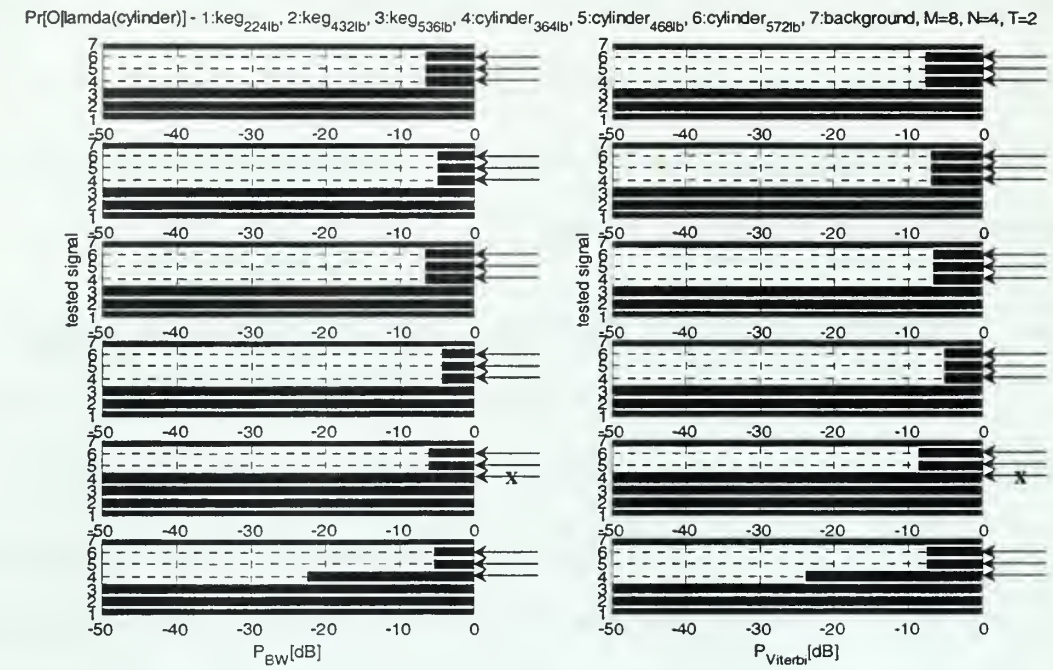
This study has only a limited number of trials available: six trials for every type of mine-like object. Thus, we used cross-validation by successively selecting five trials for training and the last one for testing. The procedure was repeated six times rotating the testing signal each time. The code implementation is given in Appendices C.5 and C.6. We found that the best performance is achieved with the number of HMM states N equal to four.

We created two ergodic HMM models: one for the keg class and one for the cylinder. Thus, we used a total number of 5(trials)x3(types of weight)=15 signals for the HMM training for every class (i.e., model). We selected an ergodic model as it performed better than the left-to-right model for the data.

F. MULTIPLE WEIGHTS SCORING AND RESULTS

The MATLAB file sequence.m given in Appendix C.6 performs all HMM training and scoring. Note that the testing signal is rotated each time, and the results

plotted, as shown in Figure 4.3a (cylinder training case) and in Figure 4.3b (keg training case).

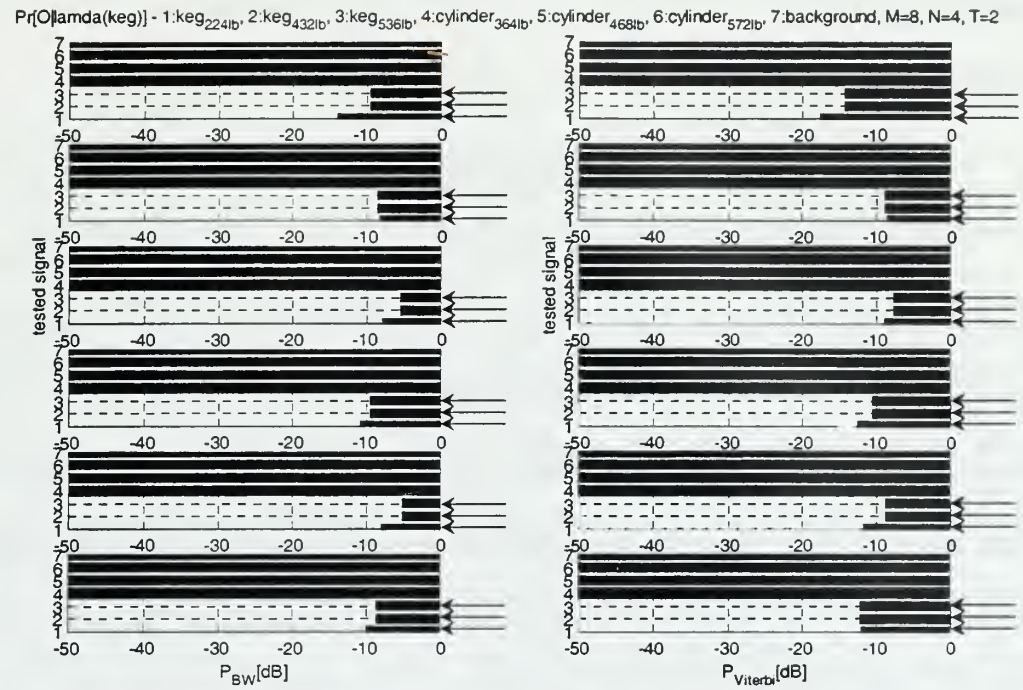


$P_{BW}[dB]$: $10\log_{10}Pr_{bw}(O|\lambda)$

$P_{Viterbi}[dB]$: $10\log_{10}Pr_{v}(O|\lambda)$

←—— : Correct Decision
←x— : Wrong Decision

Figure 4.3a HMM testing results for the cylinder model (multiple weights). $Pr_{bw}(O|\lambda_{cylinder})$ and $Pr_{v}(O|\lambda_{cylinder})$ for all testing signals and all rotations (every row). The model $\lambda_{cylinder}$ is created by training all cylinder signals (5x3=15). The decision is correct whenever the tested signals (4,5,6) have a higher Probability (i.e., closer to 0 on a dB scale) than 1, 2, 3, or 7 (keg signals and background). Each row represents one of the six iterations of the rotation between testing and training signals.



$P_{BW}[dB]$: $10\log_{10}Pr_{bw}(O|\lambda)$

$P_{Viterbi}[dB]$: $10\log_{10}Pr_{v}(O|\lambda)$

←—— : Correct Decision
←x— : Wrong Decision

Figure 4.3b HMM testing results for the keg model (multiple weights). $Pr_{bw}(O|\lambda_{keg})$ and $Pr_{v}(O|\lambda_{keg})$ for all testing signals and all rotations (every row). The model λ_{keg} is created by training all keg signals (5x3=15). The decision is correct whenever the tested signals (1, 2, 3) have a higher probability (i.e., closer to 0 on a dB scale) than 4, 5, 6, or 7 (cylinder signals and background). Each row represents one of the six iterations of the rotation between testing and training signals.

We used as background (non target) signals, the six trials obtained from the 2nd geophone during the keg experiment which didn't track any target signal. Figures 4.3a and 4.3b show that an decision error for one case only. The signal considered in that set-up is the 4th tested signal, i.e., the cylinder with weight of 368lbs. Figure 4.2 shows that it is the weakest signal of all the mine signals. The total number of testing data is

2(mines)x3(weights)x6(rotations)=36, thus the overall classification performance of the system is (36-1)/36 =97.2%.

G. MULTIPLE DISTANCES EXPERIMENT

Up to this point we showed that we can recognize the shape of the mine-like objects for a fixed distance. Next, we apply the HMM-based classifier to recognize these objects located at three different distances from the actuator, as we need to be able to detect and recognize a mine independent of the distance from the sonar equipment for a more realistic set-up. Experiments conducted on the 6th and the 10th of November 1998 provide the following data by moving the location of the geophones between the actuator and the target which stays fixed at 16ft, as illustrated in Appendix B.6. Thus, the following three set-ups are available:

- Distance between actuator and geophone equal to 6ft, resulting in the total distance (actuator-geophone-target-geophone) equal to 26ft,
- Distance between actuator and geophone equal to 8ft, resulting in the total distance (actuator-geophone-target-geophone) equal to 24ft.
- Distance between actuator and geophone equal to 10ft, resulting in the total distance (actuator-geophone-target-geophone) equal to 22ft.

Note that the reflected signals obtained for the cylinder for total distances equal to 22ft and 24ft were very weak (in fact, same range as that of the background noise). Thus, we used this data for the powder keg case only.

We created two classes again. The first class contains the keg data with weight 224lbs obtained for the 6ft, 8ft, and 10ft experimental set-ups, and 6 trials at each distance. The second class is composed of the cylinder data with weight 364lbs for the experimental set-up of 10ft. Six trials are also available for the experimental set-up. All signals were segmented again around the target location, as done before, resulting in

signals with length equal to 400 points, as shown in Figure 4.4. Again, Figure 4.4 shows that there is also a viewable consistency for the mine-like keg object. No conclusion can be drawn for the cylinder, as the signal was too weak to be usable.

We used the same feature extraction as that considered earlier for the multiple weight experiment (2 segments, decimation, VQ, number of symbols M=8), and the implementation is presented in Appendix C.9. The model selected is ergodic with four states (N=4). Cross-validation was used again due to the limited amount of data available. Scoring results obtained for the keg class model and the cylinder class model are presented in Figures 4.5a and 4.5b respectively. The MATLAB files implementations for the HMM training/testing using multiple distances case are presented in Appendices C.10, C.11, and C.12.

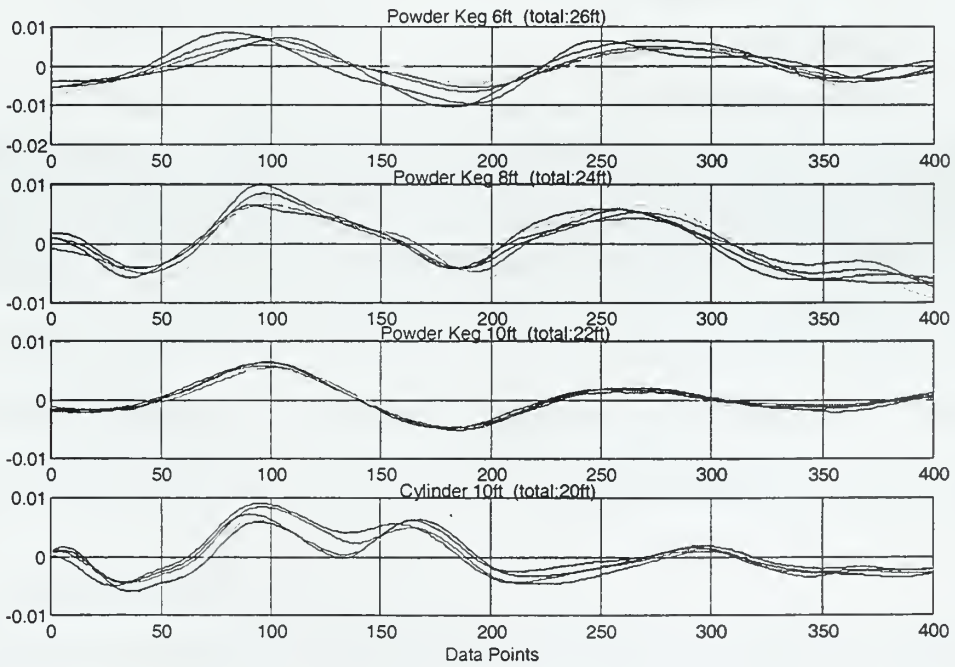
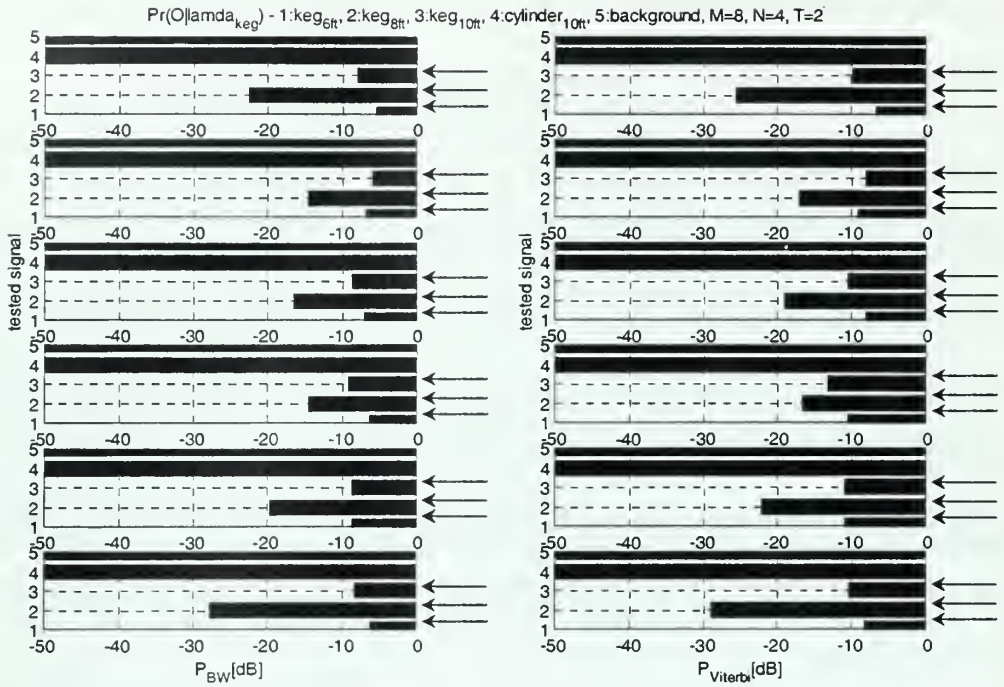


Figure 4.4 Imaginary component of the cross-power for 3 different distances for the keg target and one distance for the cylinder. Each plot illustrates 6 different trials (for the same target, and distance).

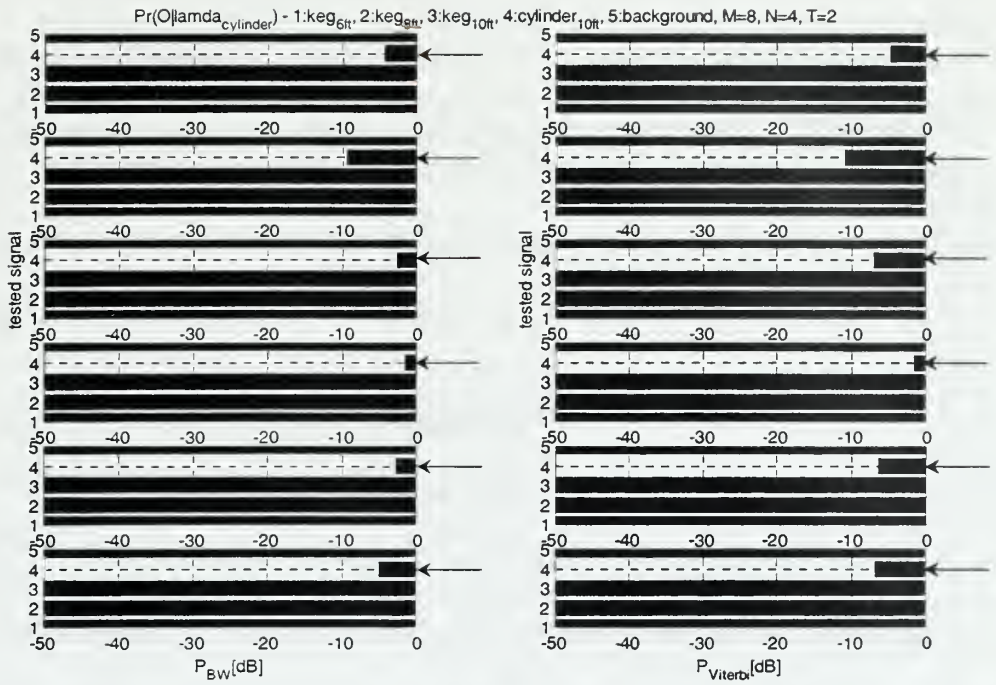


$P_{BW}[dB]$: $10\log_{10}Pr_{bw}(O|\lambda)$

$P_{Viterbi}[dB]$: $10\log_{10}Pr_{v}(O|\lambda)$

←——: Correct Decision

Figure 4.5a $Pr_{bw}(O|\lambda_{keg})$ and $Pr_{v}(O|\lambda_{keg})$ for all testing signals and all rotations (every row). The model λ_{keg} is created by training all keg signals (5x3=15). The decision is correct whenever the tested signals (1, 2, 3) have a higher probability (i.e., closer to 0 on a dB scale) than 4, or 5 (cylinder signals and background). Each row represents one of the six iterations of the rotation between testing and training signals.



P_{BW}[dB]: $10\log_{10}Pr_{bw}(O|\lambda)$

$P_{Viterbi}$[dB]: $10\log_{10}Pr_{v}(O|\lambda)$

←——: Correct Decision

Figure 4.5b $Pr_{bw}(O|\lambda_{cylinder})$ and $Pr_{v}(O|\lambda_{cylinder})$ for all testing signals and all rotations (every row). The model $\lambda_{cylinder}$ is created by training all cylinder signals (5x1=5). The decision is correct whenever the tested signals (4) have a higher probability (i.e., closer to 0 on a dB scale) than 1, 2, 3, or 5 (keg signals and background). Each row represents one of the six iterations of the rotation between testing and training signals.

Figures 4.5a and 4.5b show that the system performs 100% correct detection, which seems to indicates that the HMMs took advantage of the consistency of the signals at different distances. However, additional data is needed to make further conclusions.

H. CONCLUSIONS

This section described a HMM-based mine-like object classification system using the seismo-acoustic waves provided by the NPS project [10-12]. Initial results indicate that the HMM-based classifier can recognize the type of mine-like object, independent of

the object weight with a 97% accuracy. Results also indicate that it can recognize the object type at different distances with a 100% accuracy. However, the experiments were conducted with very few data, and further work needs to be done to confirm these initial findings by using a larger data set.

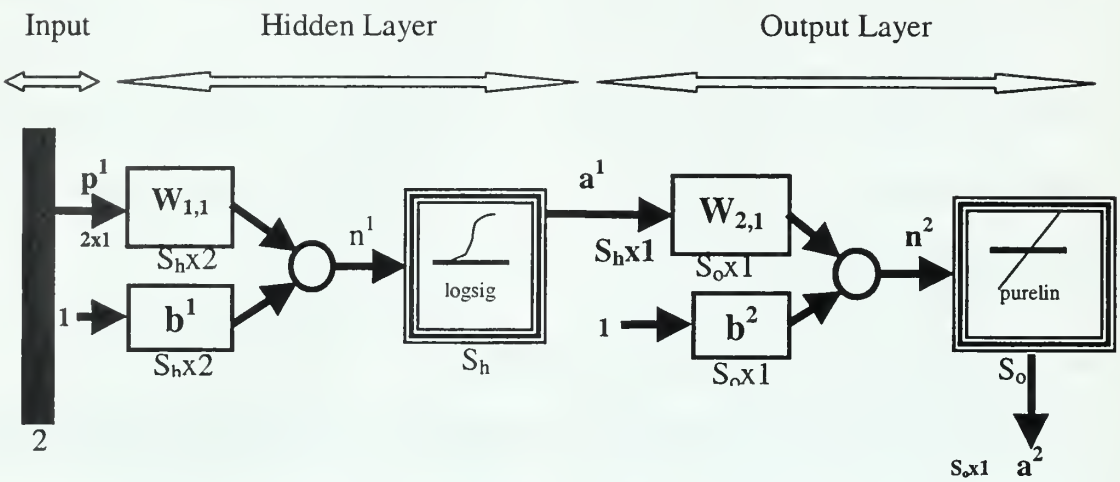
V. MINE-LIKE OBJECT RECOGNITION USING NEURAL NETWORKS

This chapter considers the application of a back-propagation neural network classifier to the same data as that considered earlier with the HMM-based classifier. The goal of this chapter is to compare the resulting performances obtained with the two different implementations.

A. NEURAL NETWORK DESCRIPTION

NN input feature vectors are those obtained after the VQ step described earlier in Chapter III. Thus, we have two signal classes again: the keg and the cylinder class. We select a back-propagation feed-forward NN (BPNN) with one hidden layer and two outputs (one for each class of mine-like objects). Note we do not specific a target output for the background signal as: 1) there is no consistency between the different background signals available, and 2) we do not use this classification technique to confirm a target existence.

The network specific structure is described in Figure 5.1. BPNNs use the steepest descent algorithm, or variants of it, to find the weights which minimize the squared error between target and network outputs [2,14]. BPNN may use one or more hidden layers of sigmoid neurons, and one output layer of linear neurons[14].



S_h: # neurons in the hidden layer (in our case Sh=60)
S_o: # neurons in the output layer (in our case, S_o=1)
$a^1=\text{logsig}(W_{1,1}p^1+b^1)$
$a^2=\text{purelin}(W_{2,1}a^1+b^2)$

Figure 5.1 Hidden and output layer of neurons of a backpropagation feedforward neural network.

B. MULTIPLE WEIGHTS SET-UP

We used the observations from the 3 different weights for each mine-like object (keg and cylinder), 5 trial each, as input vectors p to train the neural network. We used the numbers "1" and "2" as targets for the neural network, such as "1" for the keg class, and "2" for the cylinder class (see Appendix D.1 for the MATLAB code). Finally, we tested the 6^{th} trial of each signal by computing its output value to the trained NN. The test signal is recognized as a "keg" signal when the output is close to 1, and it is recognized at a "cylinder" signal when it is close to 2. Note that the NN output is not binary, so we set a range around output values 1 and 2, around which the data is said to belong to one class or the other. The specific range is set at 3% above or below the target

output values 1 and 2. The data is considered as not belonging to either class (other) when the NN output value is outside that range.

We also tested the background signal defined for the HMM-based classifier. We rotated training and testing observations six times, as done with the HMM-based classifier. Results for the multiple-weights case are given in Table 5.1. Note that the system recognizes all testing signals, except the first background signal, which is detected as a cylinder-class signal because the associated NN was equal to 1.999 (within the ±3% range of the target output value 2 associated to the cylinder-class). Thus, the overall recognition performance is 97%, as the total number of testing signals is 42 (7signals x 6 rotations).

Table 5.1 Multiple weights NN outputs/decisions

	Testing Signal						
Rotation	Keg_{224lb}	Keg_{432lb}	Keg_{536lb}	Cyl_{364lb}	Cyl_{468lb}	Cyl_{572lb}	Backgr
1	0.9999	0.9999	0.9999	1.9999	1.9999	1.9999	1.9999
	√	√	√	√	√	√	**X (Cyl)**
2	0.9831	0.9958	0.9958	1.9962	1.9780	1.9780	1.1827
	√	√	√	√	√	√	√
3	1.0043	0.9993	0.9993	1.9962	1.9962	1.9962	1.3478
	√	√	√	√	√	√	√
4	1.0001	0.9999	0.9999	1.9962	1.9996	1.9996	0.7544
	√	√	√	√	√	√	√
5	0.9999	1.0000	1.0000	1.9999	1.9999	1.9996	2.4937
	√	√	√	√	√	√	√
6	1.0117	0.9993	0.9993	1.9977	2.0053	2.0053	0.2342
	√	√	√	√	√	√	√

√: Correct Decision
X: Wrong Decision

C. MULTIPLE DISTANCES SET-UP

Similarly, the NN is used for the multiple distance set-up, as considered earlier in Section IV. Results are presented in Table 5.2, and the code is given in Appendix D.2.

Table 5.2 Multiple distances NN outputs/decisions

	Testing Signal				
Rotation	Keg_{6ft}	Keg_{8ft}	Keg_{510ft}	Cyl_{10ft}	Backgr
1	0.9966	1.3831	1.001	1.9705	1.0014
	√	**X(back)**	√	√	**X (Cyl)**
2	0.9942	0.9924	1.012	2.012	2.3552
	√	√	√	√	√
3	1.023	1.042	0.9923	1.999	1.3245
	√	√	√	√	√
4	1.034	1.045	0.9943	0.9938	1.5423
	√	√	√	√	√
5	1.003	1.000	0.9945	1.9936	1.5945
	√	√	√	√	√
6	1.010	0.9995	0.9991	1.9994	0.2342
	√	√	√	√	√

√: Correct Decision
X: Wrong Decision

Table 5.2 shows that all decisions are correct, except in two cases:

1) the first trial of the keg data at 8ft gets detected as background,

2) the first trial of the background signal gets detected as cylinder-class data.

Thus, the overall classifier performance is 93% for this set-up.

D. CONCLUSIONS/COMPARISON WITH THE HMM-BASED CLASSIFIER

These results show that the two classifiers performed in a similar manner. The overall performance was 97% for the multiple weights set-up, while the HMM-based classifier performance was 100%, and the NN 93% for the multiple distance set-up. Note that no background signal was used during the NN training, and that we assigned as background data which NN output didn't fall into the prescribed range.

Finally, we measure the speed of the execution of the MATLAB file for the evaluation of all the results for the multiple distances case. Thus, in a Pentium III 450MHz, 128MB Ram, the execution time for the HMMs was 5s, and for the NN was 15s, thus the HMM was 3 times faster that the NN (training and testing).

VI. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

This study presented an introduction to HMMs and their applications to classification problems. HMMs require the selection of consistent characteristics between the classes to perform well. In addition, many other parameters have to be carefully selected to set-up the HMM to have it perform successfully. We implemented the HMM using MATLAB and tested it on a simple four isolated word recognition problem.

The HMM-based classifier implemented in this study performed very well on the limited (two classes) mine-like object recognition experiment. Results also show that its performance is similar to that obtained with a BPNN. However, the classifier needs to be run using larger data sets to confirm the present findings.

B. RECOMMENDATIONS

The reflected signals from the targets were quite weak, especially when the target distance was large, and the mine weight low. Furthermore, the received reflected signals from the two geophones sometimes had different strengths, (see Appendix B.10), or weren't received by both geophones, (see Appendix B.11). Thus, additional data is required to further the classification/recognition process, and data collection is in progress, as of September 1999 [15]. Finally, a more complicated HMM-based set-up would probably be needed, if we want to train the classifier with different types of mines, possibly using a higher number of segments (T), and/or symbols (M), and/or states (N).

APPENDIX A. HIDDEN MARKOV MODEL MATLAB PROGRAMS

This appendix contains the various MATLAB programs used for the HMM-based classification of isolated words.

APPENDIX A1. FEATURES EXTRACTION/VECTOR QUANTIZATION USING NEURAL NETWORKS

```
% Filename: training.m
% Written by: M. Zambartas
% Date Last Modified 10 August 1999
% Purpose: Features Analysis (LPC & Energy) of 4 words (Statistics,
Microsoft, Instructor,
%          and Professor), 3 trials each + 1 test word, vector
quantization using a competitive layer neural network
%          for HMM classification use.


clear
% M: dimension of codebook, N: number of states
M=4;N=8;
rand('seed',1);
% the mat files contain the word signals
%load stat.mat;load micro.mat;load instrprof.mat;
load words.mat;
% signals interpolation
% coeff is the fuction that gives back the LCP + energy coefficients.
% T is the number of segments
[csta1,T]=coeff(sta1);[csta2,T]=coeff(sta2);[csta3,T]=coeff(sta3);[csta
4,T]=coeff(sta4);
[cmicro1,T]=coeff(micro1);[cmicro2,T]=coeff(micro2);
[cmicro3,T]=coeff(micro3);[cmicro4,T]=coeff(micro4);
[cinstr1,T]=coeff(instr1);[cinstr2,T]=coeff(instr2);
[cinstr3,T]=coeff(instr3);[cinstrtest,T]=coeff(instrtest);
[cprof1,T]=coeff(prof1);[cprof2,T]=coeff(prof2);
[cprof3,T]=coeff(prof3);[cproftest,T]=coeff(proftest);


n=1:100;
% we convert the results by a constant, because the NN works better
convert=2000;
csta1=csta1*convert;csta2=csta2*convert;csta3=csta3*convert;csta4=csta4
*convert;
cmicro1=cmicro1*convert;cmicro2=cmicro2*convert;cmicro3=cmicro3*convert
;cmicro4=cmicro4*convert;
cinstr1=cinstr1*convert;cinstr2=cinstr2*convert;cinstr3=cinstr3*convert
;cinstrtest=cinstrtest*convert;
cprof1=cprof1*convert;cprof2=cprof2*convert;cprof3=cprof3*convert;cprof
test=cproftest*convert;

% vector quantization with NN
p1=[csta1;csta2;csta3;
cmicro1;cmicro2;cmicro3;cinstr1;cinstr2;cinstr3]';
pr1=minmax(p1);

net=newc(pr1,M,1);
net.trainParam.epochs=1000;
net.trainParam.show=100;
net=train(net,p1);
w=net.IW{1};
```

```
ys=sim(net,csta1');
% classsta is the observations vector O for the first trial of the word
'Statistics'.
classsta=vec2ind(ys);
ys2=sim(net,csta2');
classsta2=vec2ind(ys2);
ys3=sim(net,csta3');
classsta3=vec2ind(ys3);
ys4=sim(net,csta4');
classstatest=vec2ind(ys4);
ym=sim(net,cmicro1');
classmicro=vec2ind(ym);
ym2=sim(net,cmicro2');
classmicro2=vec2ind(ym2);
ym3=sim(net,cmicro3');
classmicro3=vec2ind(ym3);
ym4=sim(net,cmicro4');
classmicrotest=vec2ind(ym4);

yi=sim(net,cinstr1');
classinstr=vec2ind(yi);
yi2=sim(net,cinstr2');
classinstr2=vec2ind(yi2);
yi3=sim(net,cinstr3');
classinstr3=vec2ind(yi3);
yi4=sim(net,cinstrtest');
classinstrtest=vec2ind(yi4);
yp=sim(net,cprof1');
classprof=vec2ind(yp);
yp2=sim(net,cprof2');
classprof2=vec2ind(yp2);
yp3=sim(net,cprof3');
classprof3=vec2ind(yp3);
yp4=sim(net,cproftest');
classproftest=vec2ind(yp4);


% we divide by the constant multiplied before
csta1=csta1/convert;csta2=csta2/convert;csta3=csta3/convert;csta4=csta4
/convert;w=w/convert;
cmicro1=cmicro1/convert;cmicro2=cmicro2/convert;cmicro3=cmicro3/convert
;cmicro4=cmicro4/convert;
cinstr1=cinstr1/convert;cinstr2=cinstr2/convert;cinstr3=cinstr3/convert
;cinstrtest=cinstrtest/convert;
cprof1=cprof1/convert;cprof2=cprof2/convert;cprof3=cprof3/convert;cprof
test=cproftest/convert;

% Test of vector quantization
figure(1)
subplot(4,2,1);plot(1:8,csta1(1,:),'*',1:8,w(classsta(1),:)),title(['ve
ctor1,Class=' num2str(classsta(1)) ' M=' num2str(M)])
subplot(4,2,2);plot(1:8,csta1(2,:),'*',1:8,w(classsta(2),:)),title(['ve
ctor2,Class=' num2str(classsta(2))])
subplot(4,2,3);plot(1:8,csta1(3,:),'*',1:8,w(classsta(3),:)),title(['ve
ctor3,Class=' num2str(classsta(3))])
subplot(4,2,4);plot(1:8,csta1(4,:),'*',1:8,w(classsta(4),:)),title(['ve
ctor4,Class=' num2str(classsta(4))])
```

```
subplot(4,2,5);plot(1:8,csta1(5,:),'*',1:8,w(classsta(5),:)),title(['ve
ctor5,Class=' num2str(classsta(5))])
subplot(4,2,6);plot(1:8,csta1(6,:),'*',1:8,w(classsta(6),:)),title(['ve
ctor6,Class=' num2str(classsta(6))])
subplot(4,2,7);plot(1:8,csta1(7,:),'*',1:8,w(classsta(7),:)),title(['ve
ctor7,Class=' num2str(classsta(7))])
subplot(4,2,8);plot(1:8,csta1(8,:),'*',1:8,w(classsta(8),:)),title(['ve
ctor8,Class=' num2str(classsta(8))])
figure(2)
subplot(4,2,1);plot(1:8,cmicro1(1,:),'*',1:8,w(classmicro(1),:)),title(
['vector1,Class=' num2str(classmicro(1)) ' M=' num2str(M)])
subplot(4,2,2);plot(1:8,cmicro1(2,:),'*',1:8,w(classmicro(2),:)),title(
['vector2,Class=' num2str(classmicro(2))])
subplot(4,2,3);plot(1:8,cmicro1(3,:),'*',1:8,w(classmicro(3),:)),title(
['vector3,Class=' num2str(classmicro(3))])
subplot(4,2,4);plot(1:8,cmicro1(4,:),'*',1:8,w(classmicro(4),:)),title(
['vector4,Class=' num2str(classmicro(4))])
subplot(4,2,5);plot(1:8,cmicro1(5,:),'*',1:8,w(classmicro(5),:)),title(
['vector5,Class=' num2str(classmicro(5))])
subplot(4,2,6);plot(1:8,cmicro1(6,:),'*',1:8,w(classmicro(6),:)),title(
['vector6,Class=' num2str(classmicro(6))])
subplot(4,2,7);plot(1:8,cmicro1(7,:),'*',1:8,w(classmicro(7),:)),title(
['vector7,Class=' num2str(classmicro(7))])
subplot(4,2,8);plot(1:8,cmicro1(8,:),'*',1:8,w(classmicro(8),:)),title(
['vector8,Class=' num2str(classmicro(8))])
%qa(1:T,:)=w(classmicro(1:T),:);
%qa(1:T,:)=w(classsta(1:T),:);
for n=1:T
   distances(n)= dist(csta1(n,:),w(classsta(n),:)');
   distancem(n)= dist(cmicro1(n,:),w(classmicro(n),:)');

end
figure(3)
stem(distances),title('Euclidean Distance original/quantized vectors')
figure(4)
stem(distancem),title('Euclidean Distance original/quantized vectors')
```

APPENDIX A2. FEATURES EXTRACTION/VECTOR QUANTIZATION USING K-MEANS

```
% Filename: training1.m
% Written by: M. Zambartas
% Date Last Modified 10 August 1999
% Purpose: Features Analysis (LPC & Energy) of 4 words (Statistics,
Microsoft, Instructor,
%          and Professor), 3 trials each + 1 test word, vector
quantization using k-means
%          for HMM classification use.

clear
M=4;N=8;
n_it=20;
rand('seed',1);

%load stat.mat;load micro.mat;load instrprof.mat;
load words.mat;
% signals interpolation
% T is the number of segments
% ssta1,ssta2 ... coefficients of each trial
[csta1,T]=coeff(sta1);[csta2,T]=coeff(sta2);[csta3,T]=coeff(sta3);[csta
4,T]=coeff(sta4);
[cmicro1,T]=coeff(micro1);[cmicro2,T]=coeff(micro2);
[cmicro3,T]=coeff(micro3);[cmicro4,T]=coeff(micro4);
[cinstr1,T]=coeff(instr1);[cinstr2,T]=coeff(instr2);
[cinstr3,T]=coeff(instr3);[cinstrtest,T]=coeff(instrtest);
[cprof1,T]=coeff(prof1);[cprof2,T]=coeff(prof2);
[cprof3,T]=coeff(prof3);[cproftest,T]=coeff(proftest);


% vector quantization
p1=[csta1;csta2;csta3;
cmicro1;cmicro2;cmicro3;cinstr1;cinstr2;cinstr3];
% CodeBook creation:
[CODE, label, dist] = svq(p1, M, n_it);
% Vector quantization
[w classsta, dist] = vq(csta1, CODE, n_it);
[w2, classsta2, dist] = vq(csta2, CODE, n_it);
[w3 classsta3, dist] = vq(csta3, CODE, n_it);
[wt, classstatest, dist] = vq(csta4, CODE, n_it);
[w classmicro, dist] = vq(cmicro1, CODE, n_it);
[w classmicro2, dist] = vq(cmicro2, CODE, n_it);
[w classmicro3, dist] = vq(cmicro3, CODE, n_it);
[w classmicrotest, dist] = vq(cmicro4, CODE, n_it);
[w classinstr, dist] = vq(cinstr1, CODE, n_it);
[w classinstr2, dist] = vq(cinstr2, CODE, n_it);
[w classinstr3, dist] = vq(cinstr3, CODE, n_it);
[w classinstrtest, dist] = vq(cinstrtest, CODE, n_it);
[w classprof, dist] = vq(cprof1, CODE, n_it);
[w classprof2, dist] = vq(cprof2, CODE, n_it);
[w classprof3, dist] = vq(cprof3, CODE, n_it);
[w classproftest, dist] = vq(cproftest, CODE, n_it);
```

```
% Test of vector quantization
figure(1)
subplot(4,2,1);plot(1:8,csta1(1,:),'*',1:8,w(1)),title(['vector1,Class=
' num2str(classsta(1)) ' M=' num2str(M)])
subplot(4,2,2);plot(1:8,csta1(2,:),'*',1:8,w(2)),title(['vector2,Class=
' num2str(classsta(2))])
subplot(4,2,3);plot(1:8,csta1(3,:),'*',1:8,w(3)),title(['vector3,Class=
' num2str(classsta(3))])
subplot(4,2,4);plot(1:8,csta1(4,:),'*',1:8,w(4)),title(['vector4,Class=
' num2str(classsta(4))])
subplot(4,2,5);plot(1:8,csta1(5,:),'*',1:8,w(5)),title(['vector5,Class=
' num2str(classsta(5))])
subplot(4,2,6);plot(1:8,csta1(6,:),'*',1:8,w(6)),title(['vector6,Class=
' num2str(classsta(6))])
subplot(4,2,7);plot(1:8,csta1(7,:),'*',1:8,w(7)),title(['vector7,Class=
' num2str(classsta(7))])
subplot(4,2,8);plot(1:8,csta1(8,:),'*',1:8,w(8)),title(['vector8,Class=
' num2str(classsta(8))])
figure(2)
subplot(4,2,1);plot(1:8,cmicro1(1,:),'*',1:8,w(classmicro(1),:)),title(
['vector1,Class=' num2str(classmicro(1)) ' M=' num2str(M)])
subplot(4,2,2);plot(1:8,cmicro1(2,:),'*',1:8,w(classmicro(2),:)),title(
['vector2,Class=' num2str(classmicro(2))])
subplot(4,2,3);plot(1:8,cmicro1(3,:),'*',1:8,w(classmicro(3),:)),title(
['vector3,Class=' num2str(classmicro(3))])
subplot(4,2,4);plot(1:8,cmicro1(4,:),'*',1:8,w(classmicro(4),:)),title(
['vector4,Class=' num2str(classmicro(4))])
subplot(4,2,5);plot(1:8,cmicro1(5,:),'*',1:8,w(classmicro(5),:)),title(
['vector5,Class=' num2str(classmicro(5))])
subplot(4,2,6);plot(1:8,cmicro1(6,:),'*',1:8,w(classmicro(6),:)),title(
['vector6,Class=' num2str(classmicro(6))])
subplot(4,2,7);plot(1:8,cmicro1(7,:),'*',1:8,w(classmicro(7),:)),title(
['vector7,Class=' num2str(classmicro(7))])
subplot(4,2,8);plot(1:8,cmicro1(8,:),'*',1:8,w(classmicro(8),:)),title(
['vector8,Class=' num2str(classmicro(8))])
%qa(1:T,:)=w(classmicro(1:T),:);
%qa(1:T,:)=w(classsta(1:T),:);
for n=1:T
   distances(n)= dist(csta1(n,:),w(classsta(n),:)');
   distancem(n)= dist(cmicro1(n,:),w(classmicro(n),:)');

end
figure(3)
stem(distances),title('Euclidean Distance original/quantized vectors')
figure(4)
stem(distancem),title('Euclidean Distance original/quantized vectors')
```

APPENDIX A3. FEATURES EXTRACTION (LPC + ENERGY) FUNCTION

```
% Filename: coeff.m
% Written by: M. Zambartas
% Date Last Modified 10 August 1999
% Purpose: fuction that computes the LPC coefficients and the energy of
each segment
%  T:  total number of segments
%  c:  Coefficients vector

function [c,T]=coeff(word);
% signals interpolation
%word=word-mean(word);
l=length(word);
% interpolation of word, so every word has 10000 data points
int=interp1(1:l,word,1:l/10000:10000);
% Pre-emphasis filtering:
pre=filter([1,-.98],1,int);

start=1;
step=1200;
l=10000-step;
% Overlaping of segments:
overlap=round((10/100)*step);
%# of Observations T:
T=l/step;
T=floor(T);
for n=1:T
   c(n,1:8)=ar_covar(pre(start:start+step-1+overlap),7);
   start=start+step;
   x=xcorr(pre(start:start+step-1+overlap));
   lx=length(x);
   center=find(x==max(x));
   c(n,1)=x(center);
end
c(:,1)=c(:,1)/max(c(:,1));
c(:,1)=c(:,1)-.5;
```

APPENDIX A4. FORWARD VARIABLE ALGORITHM IMPLEMENTATION

```
% Filename: forward.m
% Written by: M. Zambartas
% Date Last Modified 10 August 1999
% Purpose: Implements the forward algorithm (Rabiner 1986 pg9)

function afw=forward(a,b,pi,O,N,T)
%  O:  the observation sequence
%  T:  total number of segments
%  a:  the state transition probabilities (N x N)
%  b:  observation probability distribution (N x M)
%  pi: initial state distribution (N x 1)
%  N:  total number of states
%  M:  total number of possible observation symbols (dimension of
codebook)
%  afw:  the forward variable  (N x T)
%  bbw   the backward variable (N x T)

% initial values:
afw=zeros(T,N);
afw(1,1:N)=pi(1:N).*b(1:N,O(1))';
% recursion:
% we are using eps (for zero results) to avoid divided by zero problems
if norm(afw(1,:))<eps
   afw(1,:)=eps;
end

% FW algorithm:
   for t=1:T-1
      for j=1:N

     S=0;
     for i=1:N
      % summation S:
      S=S+afw(t,i)*a(i,j);
     end
     %logafw=log10(S)+log10(b(j,O(t+1)));

     afw(t+1,j)=S*b(j,O(t+1));
       if norm(afw(t+1,j))<eps
        afw(t+1,j)=eps;
       end

     %afw(t+1,j)=10^logafw;
  end
end
```

APPENDIX A5. BACKWARD VARIABLE ALGORITHM IMPLEMENTATION

```
% Filename: backward.m
% Written by: M. Zambartas
% Date Last Modified 10 August 1999
% Purpose: Implements backward algorithm (Rabiner 1986 pg9)

function bbw=backward(a,b,pi,O,N,T)

%  O:  the observation sequence
%  T:  total number of segments
%  a:  the state transition probabilities (N x N)
%  b:  observation probability distribution (N x M)
%  pi: initial state distribution (N x 1)
%  N:  total number of states
%  M:  total number of possible observation symbols (dimension of
codebook)
%  afw:  the forward variable  (N x T)
%  bbw   the backward variable (N x T)

% BW Algorithm:
% Initial Values:

bbw=zeros(T,N);
bbw(T,1:N)=1;
% recursion:
% we are using eps (for zero results) to avoid divided by zero problems
if norm(bbw(T,:))<eps
   bbw(T,:)=eps;
end

   for t=T-1:-1:1
      for i=1:N

      S=0;
      for j=1:N
         S=S+a(i,j)*b(j,O(t+1))*bbw(t+1,j);
      end
      bbw(t,i)=S;
      if S<eps
         bbw(t,i)=eps;
      end

   end
end
```

APPENDIX A6. FW/BW VARIABLES SCALING

```
% Filename: scale.m
% Written by: M. Zambartas
% Date Last Modified 10 August 1999
% Purpose: Implements scaling of FW and BW variables, according to
Rabiner(1989 pg 272)
% We perform scaling to avoid exceeding the precision range of the
computer
%  O:  the observation sequence
%  T:  total number of segments
%  a:  the state transition probabilities (N x N)
%  b:  observation probability distribution (N x M)
%  pi: initial state distribution (N x 1)
%  N:  total number of states
%  M:  total number of possible observation symbols (dimension of
codebook)
%  afw:  the forward variable  (N x T)
%  bbw   the backward variable (N x T)

function [afw1,bbw1,c]=scale(a,b,pi,O);

N=length(a);
s=size(O);
T=s(2);
afw=forward(a,b,pi,O,N,T);
bbw=backward(a,b,pi,O,N,T);
afw2=zeros(T,N);
afw2(1,:)=afw(1,:);

c(1)=1/sum(afw(1,:));
afw1(1,1:N)=c(1)*afw(1,1:N);

for t=2:T
  for i=1:N
     S=0;
     for j=1:N
      % summation S:
      S=S+afw1(t-1,j)*a(j,i);
     end
     afw2(t,i)=S*b(i,O(t));

  end
  su=sum(afw2(t,:));
  if su==0
     su=eps;
  end
  c(t)=1/su;
  afw1(t,:)=c(t)*afw2(t,:);
  end
bbw1=zeros(T,N);
bbw1(T,:)=c(T);
   for t=T-1:-1:1
      for i=1:N
      S=0;
```

```
    for j=1:N
       S=S+a(i,j)*b(j,O(t+1))*bbw1(t+1,j);
    end
    bbw1(t,i)=S*c(t);
    end
end
```

APPENDIX A7. MULTIPLE OBSERVATION SEQUENCE

```
% Filename: newmodel.m
% Written by: M. Zambartas
% Date Last Modified 10 August 1999
% Purpose: Builds the O_multi matrix, which contains all trials
obsrvations and
% calls trainmulti.m function
%  T:  total number of segments
%  a:  the state transition probabilities (N x N)
%  b:  observation probability distribution (N x M)
%  pi: initial state distribution (N x 1)
%  N:  total number of states
%  M:  total number of possible observation symbols (dimension of
codebook)


% models re-estimation

% initial conditions:


O_multi=[classmicro;classmicro2;classmicro3];
%O_multi=[classsta;classsta2;classsta3];
%O_multi=[classinstr;classinstr2;classinstr3];
%O_multi=[classprof;classprof2;classprof3];

 [a,b,pi] = trainmulti(O_multi,N,T,M);
```

APPENDIX A8. BAUM-WELCH ALGORITHM (MODEL REESTIMATION)

```
% Filename: trainmodel.m
% Written by: M. Zambartas
% Date Last Modified 10 August 1999
% Purpose: this function implements the Baum-Welch reetimation
algorithm as discribed
%          in Rabiner (1986) pg 11
%  O:  the observation sequence
%  T:  total number of segments
%  a:  the state transition probabilities (N x N)
%  b:  observation probability distribution (N x M)
%  pi: initial state distribution (N x 1)
%  N:  total number of states
%  M:  total number of possible observation symbols (dimension of
codebook)
%  afw:  the forward variable  (N x T)
%  bbw   the backward variable (N x T)
%  afw1: Scaled forward variable  (N x T)
%  bbw1: Scaled backward variable (N x T)

function [a,b,pi,P] = trainmodel(O,N,T,M)
% if the model is ergotic keep a=rand(N).  If left-right then
a=triu(a), because it has
% to be upper triangular, in order to converge to an upper triangular
matrix

% Initial estimation of model lamda=(a,b,pi):
%a=rand(N);
b=rand(N,M);
a=triu(a);
% we need the summation or all rows of a and b to be 1 , so:
b = b./((sum(b.')).'*ones(1,M));
a = a./((sum(a.')).'*ones(1,N));
for j=1:N;
   pi(j)=1/(50*j);
end

% we need the summation of pi to be 1:
pi=pi./sum(pi);
for times=1:20

afw=forward(a,b,pi,O,N,T);
bbw=backward(a,b,pi,O,N,T);
% Scaling: (Rabiner 89 pg272)
% afw scaling:
a_old=a;
b_old=b;
pi_old=pi;

afw2=zeros(T,N);
afw2(1,:)=afw(1,:);

c(1)=1/sum(afw(1,:));
afw1(1,1:N)=c(1)*afw(1,1:N);
```

```
for t=2:T
  for i=1:N
     S=0;
     for j=1:N
      % summation S:
      S=S+afw1(t-1,j)*a(j,i);
     end
     afw2(t,i)=S*b(i,O(t));

  end
  su=sum(afw2(t,:));
  % to avoid 'divided by zero' errors:
  if su==0
     su=eps;
  end
  c(t)=1/su;
  afw1(t,:)=c(t)*afw2(t,:);
end
% bbw scaling:
bbw1=zeros(T,N);
bbw1(T,:)=c(T);
   for t=T-1:-1:1
      for i=1:N
      S=0;
      for j=1:N
         S=S+a(i,j)*b(j,O(t+1))*bbw1(t+1,j);
      end
      bbw1(t,i)=S*c(t);
      end
   end

% a reestimation
for i=1:N
  S2=[];
  for t=1:T-1
     S1=[];
     for j=1:N
        S1=[S1; afw1(t,i)*a(i,j)*bbw1(t+1,j)*b(j,O(t+1))];
     end
  S2=[S2 sum(S1)];
  end
  dena(i)=sum(sum(S2));
end

   for i=1:N
    for j=1:N
       S1=0;S2=0;
     for t=1:T-1
       S1=[S1; afw1(t,i)*a(i,j)*bbw1(t+1,j)*b(j,O(t+1))];

     end
     numa(i,j)=sum(S1);
     if dena(i)==0
        dena(i)=eps;
     end
     a(i,j)=numa(i,j)/dena(i);
```

```
    end
   end
   if norm(a-a_old)<.01

    a=a_old;
   end
% b reestimation
   for j=1:N
      for k=1:M
         S1=0;S2=0;
         for t=1:T
            S1=S1+afw1(t,j)*bbw1(t,j)*(O(t)==k);;
            S2=S2+afw1(t,j)*bbw1(t,j);
            if S2==0
               S2=eps;
            end
            b(j,k)=S1/S2;
         end
      end
   end
   if norm(b-b_old)<.01

    b=b_old;
   end

% P_bw(O|lamda) probability
 logP=-sum(log10(c));
 P=10^logP;
 pi(:)=(afw1(1,:).*bbw(1,:)/c(1))/P;
 %logpi=(log10(afw1(1,i))+log10(bbw(1,i))-log10(c(1)))-logP;
 %pi(i)=10^logpi;
 if norm(pi-pi_old)<.01

   pi=pi_old;
end
%fprintf('%g %g %g \n',norm(a-a_old),norm(b-b_old),norm(pi-pi_old))
if norm((a-a_old)==0)&(norm(b-b_old)==0)&(norm(pi-pi_old)==0)
   break
end
end
```

APPENDIX A9. BAUM-WELCH ALGORITHM (MODEL REESTIMATION) WITH MULTIPLE OBSERVATIONS

```
% Filename: trainmulti.m
% Written by: M. Zambartas
% Date Last Modified 10 August 1999
% Purpose: this function (re)estimates the model (a,b,lamda) for
multiple Observation
% sequence according to Rabiner (89) pg 273
% Each row of O_multi is each Observation O, where:
%  O:  the observation sequence
%  T:  total number of segments
%  a:  the state transition probabilities (N x N)
%  b:  observation probability distribution (N x M)
%  pi: initial state distribution (N x 1)
%  N:  total number of states
%  M:  total number of possible observation symbols (dimension of
codebook)
%  afw:  the forward variable  (N x T)
%  bbw   the backward variable (N x T)

function [a,b,pi] = trainmulti(O_multi,N,T,M)
% s=size(O_multi);
trials=s(1);
afw1_multi=[];bbw1_multi=[];
% Computing P(O,lamda) & scaled fw and bw probabilities for each Obs
(trial)
for k=1:trials
   [a,b,pi,P] = trainmodel(O_multi(k,:),N,T,M);
   P_multi(k)=P;
   [afw1,bbw1,c]=scale(a,b,pi,O_multi(k,:));
   afw1_multi=[afw1_multi afw1];
   bbw1_multi=[bbw1_multi bbw1];
end
% Model's re-estimation using an alternative formula:
%P_multi(1)=1;
%a_old=a;
%  for i=1:N
%   for j=1:N
%      S1k=[];S2k=[];
%       for k=1:trials
%        S1=[];S2=[];
%        for t=1:T-1
%         S1=[S1 afw1_multi(t,i+(k-
1)*N)*a_old(i,j)*b(j,O_multi(k,t+1))*bbw1_multi(t+1,j+(k-1)*N)];
%         S2=[S2 afw1_multi(t,i+(k-1)*N)*bbw1_multi(t,i+(k-1)*N)];
%        end
%       S1=sum(S1)/P_multi(k);
%       S2=sum(S2)/P_multi(k);
%       S1k=[S1k S1];
%       S2k=[S2k S2];
%       end
%      S1k=sum(S1k);S2k=sum(S2k);
%      a(i,j)=S1k/S2k;
%   end
```

```
%end

% a denominator  evaluation:
for i=1:N
 Sk=[];
 for k=1:trials
   S2=[];
  for t=1:T-1
     S1=[];
     for j=1:N
        S1=[S1; afw1_multi(t,i+(k-1)*N)*a(i,j)*bbw1_multi(t+1,j+(k-
1)*N)*b(j,O_multi(k,t+1))];
     end
  S2=[S2 sum(S1)];
  end
 Sk=[Sk S2/P_multi(k)];

 end
 dena(i)=sum(Sk);
end

% a final Re-estimation
 for i=1:N
    for j=1:N
   Sk=[];
   for k=1:trials
       S1=[];
     for t=1:T-1
       S1=[S1 afw1_multi(t,i+(k-1)*N)*a(i,j)*bbw1_multi(t+1,j+(k-
1)*N)*b(j,O_multi(k,t+1))];

     end
     Sk=[Sk sum(S1)/P_multi(k)];

  end
  a(i,j)=sum(Sk)/dena(i);
   end
end
% b reestimation:
for j=1:N
      for l=1:M
         S1k=[];S2k=[];
         for k=1:trials
         S1=[];S2=[];
         for t=1:T
            S1=[S1 afw1_multi(t,j+(k-1)*N)*bbw1_multi(t,j+(k-
1)*N)*(O_multi(k,t)==l)];
            S2=[S2 afw1_multi(t,j+(k-1)*N)*bbw1_multi(t,j+(k-1)*N)];

         end
         S1=sum(S1)/P_multi(k);S2=sum(S2)/P_multi(k);
         S1k=[S1k S1];S2k=[S2k S2];
         end
      S1k=sum(S1k);S2k=sum(S2k);
      b(j,l)=S1k/S2k;
     end
  end
```

APPENDIX A10. VITERBI ALGORITHM

```
% Filename: viterbi.m
% Written by: M. Zambartas
% Date Last Modified 10 August 1999
% Purpose: function that implements the Viterbi algorithm as discribed
in
%          Rabiner (1986) pg 11
%  O:  the observation sequence of testing word
%  T:  total number of segments
%  a:  the state transition probabilities (N x N)
%  b:  observation probability distribution (N x M)
%  pi: initial state distribution (N x 1)
%  N:  total number of states
%  M:  total number of possible observation symbols (dimension of
codebook)
%  afw:  the forward variable  (N x T)
%  bbw   the backward variable (N x T)



function [fi,mi,s_seq]=viterbi(a,b,pi,O,N,T)
% Viterbi algorithm:
% initial value fi1(i)=pi(i)*bi(O1)
fi=zeros(T,N);
fi(1,1:N)=pi(1:N).*b(1:N,O(1))';
% backtracking pointer mi:
mi=zeros(1,N);
for t=2:T
   for j=1:N
        M=[];
      for i=1:N
         M=[M fi(t-1,i)*a(i,j)];
      end
      ArgMax=find(M==max(M));
      fi(t,j)=max(M).*b(j,O(t));
      mi(t,j)=ArgMax(1);
   end
end
% Path (state sequence) backtracking:
s_seq=zeros(1,T);
% final state @ T time:
u=find(fi(T,:)==max(fi(T,:)));
s_seq(T)=u(1);
for t=T-1:-1:1
   s_seq(t)=mi(t+1,s_seq(t+1));
end
```

APPENDIX A11. SCORING (CLASSIFICATION)

```
% Filename: score.m
% Written by: M. Zambartas
% Date Last Modified 10 August 1999
% Purpose: Evaluates P_bw(O|lamda) and P_viterbi(O|lamda)
%  O:  the observation sequence of tested word
%  T:  total number of segments
%  a:  the state transition probabilities (N x N)
%  b:  observation probability distribution (N x M)
%  pi: initial state distribution (N x 1)
%  N:  total number of states
%  M:  total number of possible observation symbols (dimension of
codebook)
%  afw:  the forward variable  (N x T)
%  bbw   the backward variable (N x T)

function [P_bw,P_v]=score(a,b,pi,O)
% Scoring of an Observation
N=length(a);
s=size(b);
T=length(O);
% Viterbi probability:
[fi,mi,s_seq]=viterbi(a,b,pi,O,N,T);
P_v=max(fi(T,:));

% Baum-Welch probability:
afw=forward(a,b,pi,O,N,T);
bbw=backward(a,b,pi,O,N,T);
[afw1,bbw1,c]=scale(a,b,pi,O);
% Rabiner 89 pg 273 P(o,lamda)=1/prod(c)
sc=-sum(log10(c));
P_bw=1/prod(c);
if P_bw==NaN
   P_bw=0;
end
```

APPENDIX A12. CLASSIFICATION RESULTS PLOTS

```
% Filename: scoretest.m
% Written by: M. Zambartas
% Date Last Modified 10 August 1999
% Purpose: Plots classification results for all words, evaluates
P_bw(O|lamda) and P_viterbi(O|lamda)
% for every model
%  O:  the observation sequence of tested word
%  T:  total number of segments
%  a:  the state transition probabilities (N x N)
%  b:  observation probability distribution (N x M)
%  pi: initial state distribution (N x 1)
%  N:  total number of states
%  M:  total number of possible observation symbols (dimension of
codebook)
%  afw:  the forward variable  (N x T)
%  bbw   the backward variable (N x T)

O=classstatest;
'test: statistics'
[P_bw1,P_v1]=score(a,b,pi,O)
if (P_bw1==0)
   P_bw1=eps;
end
if (P_v1==0)
   P_v1=eps;
end

O=classmicrotest;
'test: microsoft'
[P_bw2,P_v2]=score(a,b,pi,O)
if (P_bw2==0)
   P_bw2=eps;
end
if (P_v2==0)
   P_v2=eps;
end

O=classinstrtest;
'test: instructor'

[P_bw3,P_v3]=score(a,b,pi,O)
if (P_bw3==0)
   P_bw3=eps;
end
if (P_v3==0)
   P_v3=eps;
end

O=classproftest;
'test: professor'
[P_bw4,P_v4]=score(a,b,pi,O)
if (P_bw4==0)
   P_bw4=eps;
end
```

```
if (P_v4==0)
   P_v4=eps;
end

subplot(2,1,1), barh(10*log10([P_bw1 P_bw2 P_bw3 P_bw4]));
title(['Baum-Welch Probability - 1:Statistics, 2:Microsoft,
3:Instructor 4:Professor, M=' num2str(M) ', N=' num2str(N)]),
xlabel('P_B_W[dB]'), ylabel('tested word')
grid
subplot(2,1,2), barh(10*log10([P_v1 P_v2 P_v3 P_v4]));
title(['Viterbi Probability  - 1:Statistics, 2:Microsoft, 3:Instructor
4:Professor, M=' num2str(M) ', N=' num2str(N)]),
xlabel('P_V_i_t_e_r_b_i[dB]'), ylabel('tested word')
grid
```

APPENDIX A13. VECTOR QUANTIZATION ALGORITHM

```
function [CODE, label, dist] = svq(X, lev, n_it);

%svq    Vector quantization using successive binary splitting steps.
%  Use: [CODE,label,dist] = svq(X,lev,n_it).
%  The final codebook dimension lev should be a power of two. dist
%  returns the distorsion values at the end of intermediate step.
%  n_it is the number of iterations performed in each step.

% Version 1.3
% Olivier Cappι, 28/09/94 - 04/03/97
% ENST Dpt. Signal / CNRS URA 820, Paris

% Needed functions
if (exist('vq') ~= 2)
  error('Function vq is missing.');
end;
% Turn verbose mode off
QUIET = 1;
% Input agruments
error(nargchk(3, 3, nargin));
% Dimension of imput data
[n,p] = size(X);
% Number of spliting steps
nbs = round(log2(lev));
lev = 2^nbs;
% Fixed perturbation
perturb = 0.01;

% Initialize first centroid with global mean
CODE = zeros(lev, p);
CODE_ = zeros(lev, p);
CODE(1,:) = mean(X);
label = ones(n,1);

for i=1:nbs
  % 1. Codebook splitting
  for j=1:(2^(i-1))
    CODE_(2*j-1,:) = (1+perturb) * CODE(j,:);
    CODE_(2*j,:)   = (1-perturb) * CODE(j,:);
  end;
  % 2. K-means optimization
  [CODE(1:2^i,:),label,vdist] = vq(X,CODE_(1:2^i,:),n_it,QUIET);
  dist(i) = vdist(n_it);
  fprintf(1, 'Codebook size %d:\t%.3f\n',2^i,dist(i));
end;
```

APPENDIX A14. VECTOR QUANTIZATION USING LGB ALGORITHM

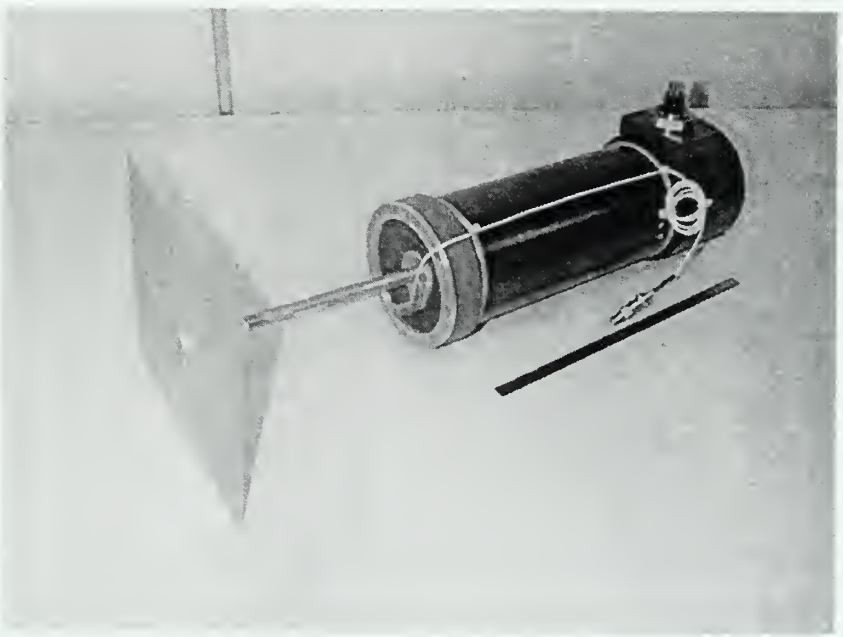
```
function [CODE_n, label, dist] = vq(X, CODE, n_it, QUIET);

%vq        Vector quantization using the K-means (or LBG) algorithm.
%       Use: [CODE_n,label,dist] = vq(X,CODE,n_it)
%  Performs n_it iterations of the K-means algorithm on X, using
%  CODE as initial codebook.

% Version 1.3
% Olivier Cappé, 28/09/94 - 16/07/97
% ENST Dpt. Signal / CNRS URA 820, Paris

error(nargchk(3, 4, nargin));
if (nargin < 4)
  QUIET = 0;
end;

% Dimensions of X
[n,p] = size(X);
% Codebook size
m  = length(CODE(:,1));
% Initialialize label array
label = zeros(1,n);
% As well as distortion values
dist = zeros(1,n_it);

% Main loop
CODE_n = CODE;
for iter = 1:n_it
  % 1. Find nearest neighbor for the squared distortion
  DIST = zeros(m,n);
  if (p > 1)
    for i = 1:m
      DIST(i,:) = sum(((X - ones(n,1)*CODE_n(i,:))').^2);
    end;
  else
    % Beware of sum when p = 1 (!)
    DIST = (ones(m,1)*X' - CODE_n*ones(1,n)).^2;
  end;
  [vm,label] = min(DIST);
  % Mean distortion
  dist(iter) = mean(vm);
  % 2. Update the codebook
  n_out = 0;
  for i = 1:m
    ind = (1:n);
    ind = ind((label == i));
    if (length(ind) == 0)
      % Isolated centroid are not modified
      n_out = n_out + 1;
    elseif (length(ind) == 1)
      % When there is only one nearest neighbor for a given codebook
entry
      CODE_n(i,:) = X(ind,:);
    else
```

```
      CODE_n(i,:) = mean(X(ind,:));
    end;
  end;
  % Affichage
  if (~QUIET)
  %  fprintf(1,'Iteration %d:\t%.3f\n',iter,dist(iter));
  end;
  if (n_out > 0)
  %  fprintf(1,'  Warning : %.0f isolated centroids\n',n_out);
  end;
end;
```

APPENDIX B. SEISMO-ACOUSTIC SONAR PROJECT INFORMATION

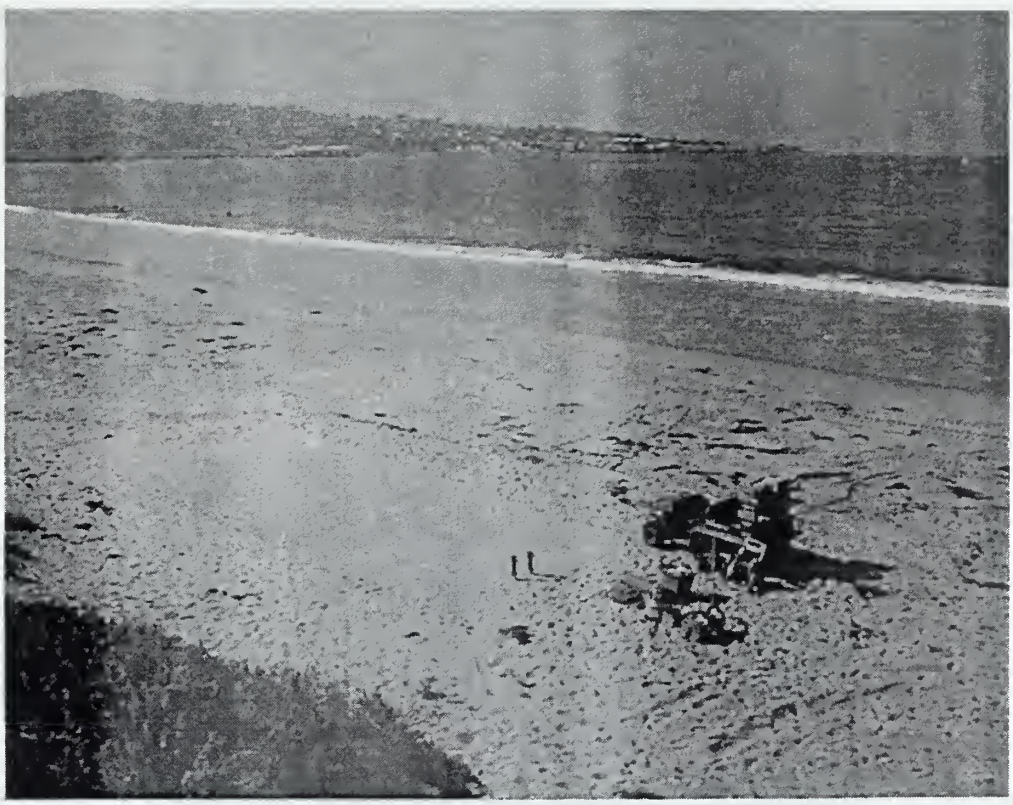
This appendix contains some basic information regarding the NPS seismo-acoustic sonar project. Further details may be found in [10, 11, 12].

APPENDIX B1. SEISMIC WAVE ACTUATOR



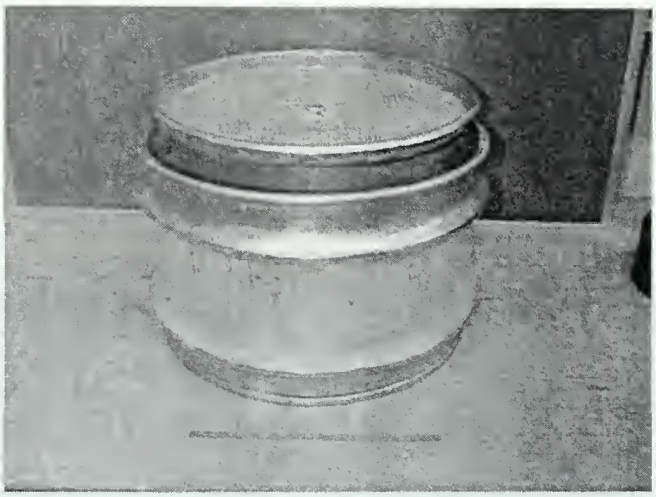
Actuator with waterproof case and coupling device [12]

APPENDIX B2. BEACH TEST SITE

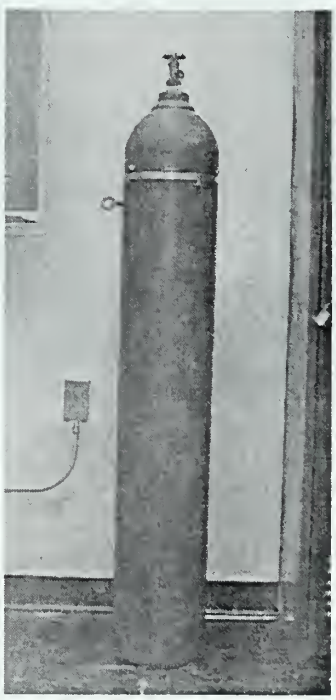


Beach test site with data collection equipment [12]

APPENDIX B3. MINE-LIKE OBJECTS

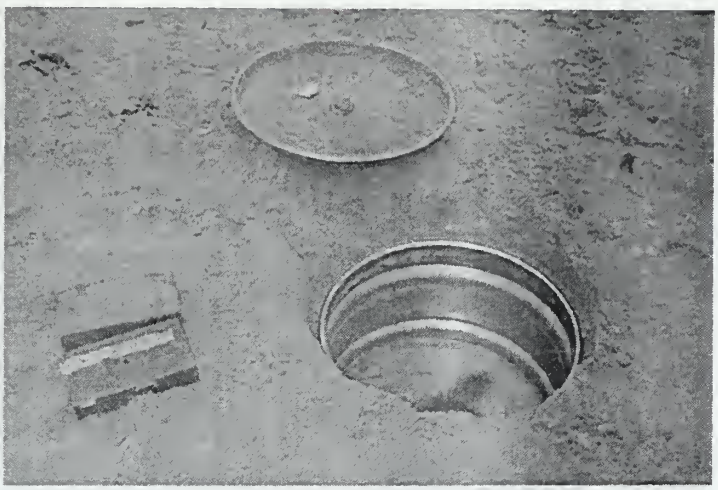


Powder keg target with lid open to show access [12]

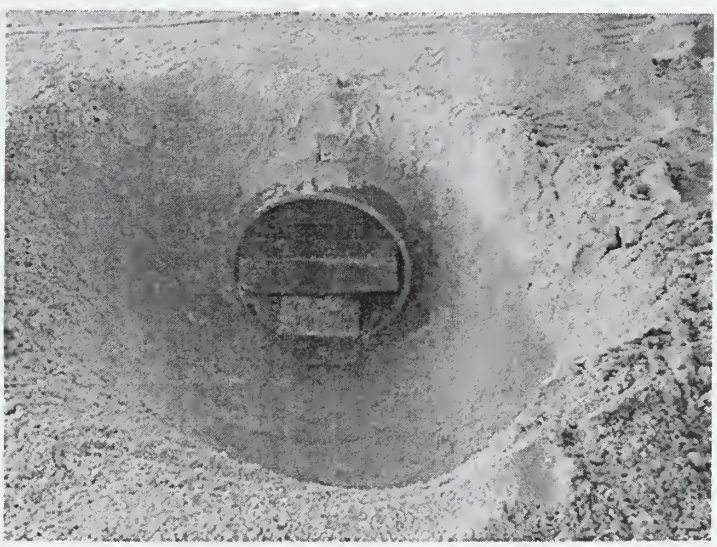


Gas cylinder target [12]

APPENDIX B4. BURIED MINE-LIKE OBJECTS

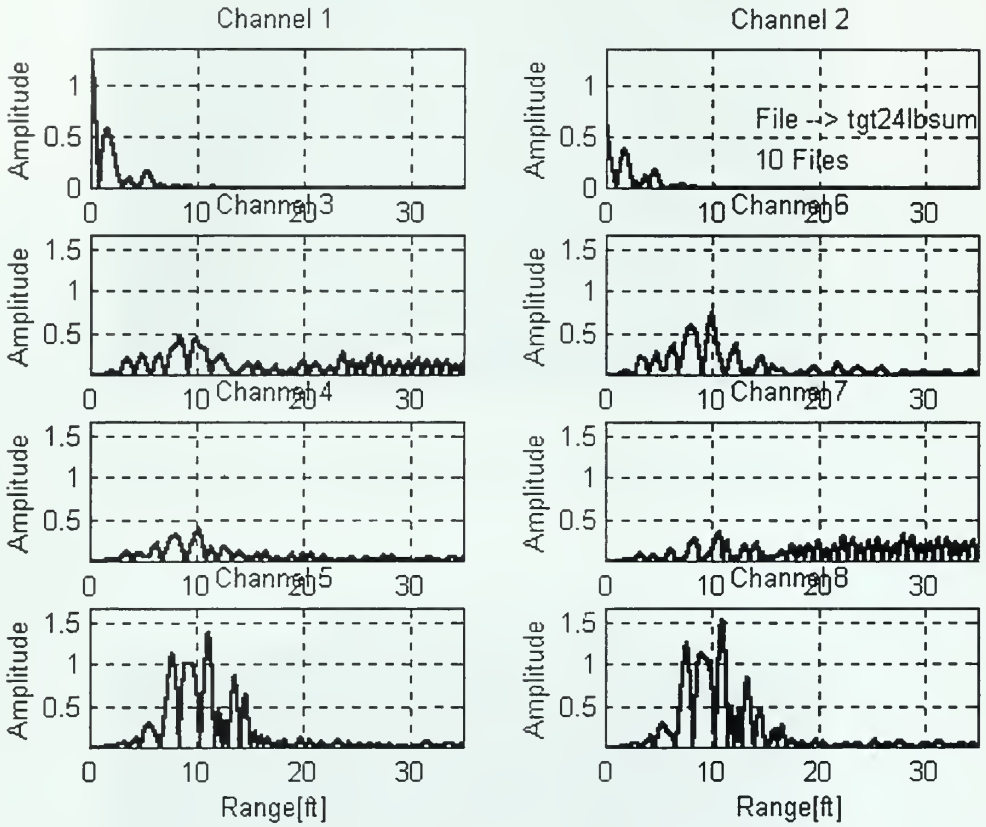


Buried powder keg target with top removed [12]



Buried cylinder with end cap removed [12]

APPENDIX B5. 8-CHANNEL DATA PLOT [12]



8-Channel data plot of the received signals. Channels 1 and 2 come from the accelerator meters. Channels 3, 4, 5 represent the x, y, z motion of the 1st geophone, and channels 6, 7, 8 the x, y, z motion of the 2nd geophone.

APPENDIX B6. TEST SETUP FOR HELIUM GAS TANK AND GUNPOWDER KEG TARGET TESTS (INCREASING MASS) [12]

This appendix contains the experimental setup for target detection with increasing mass tests conducted on November 6, 1998 and November 10, 1998.

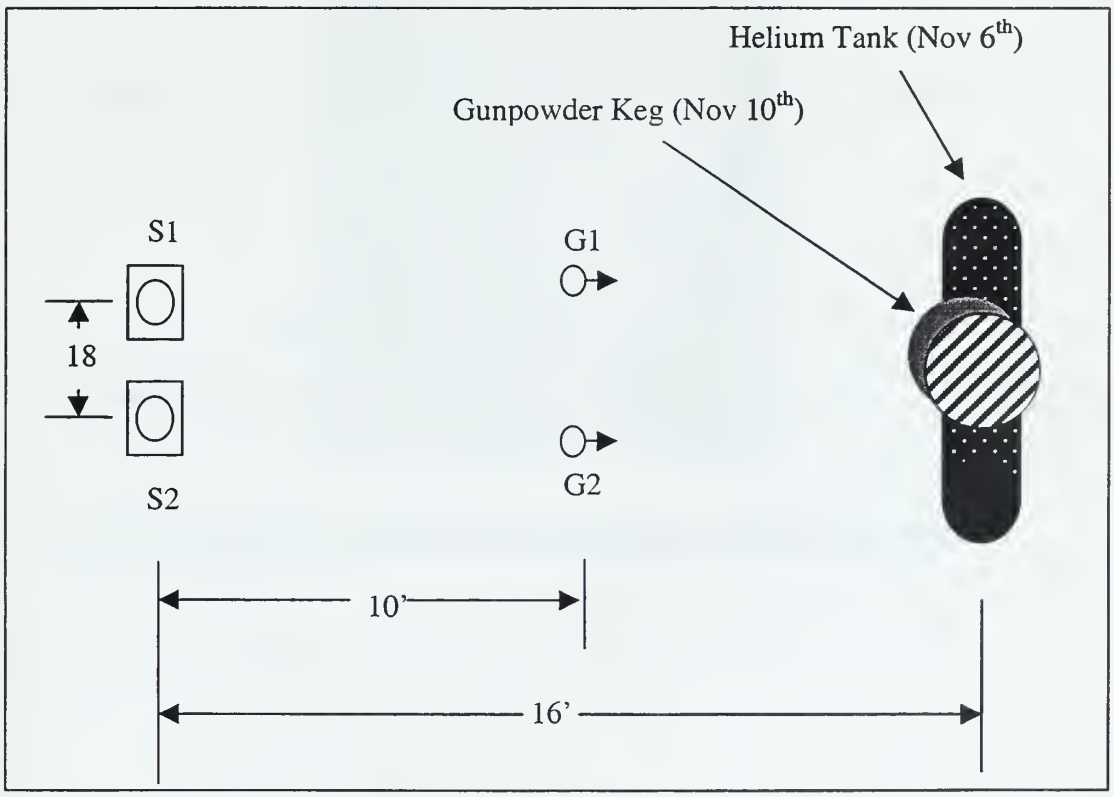
Source #1

Orientation: Vertical
Voltage: 10V (20Vpk-pk)
Frequency: 80Hz
Cycles: 1
Geophone #1
Filter: High Pass 40Hz
Gain: 40db

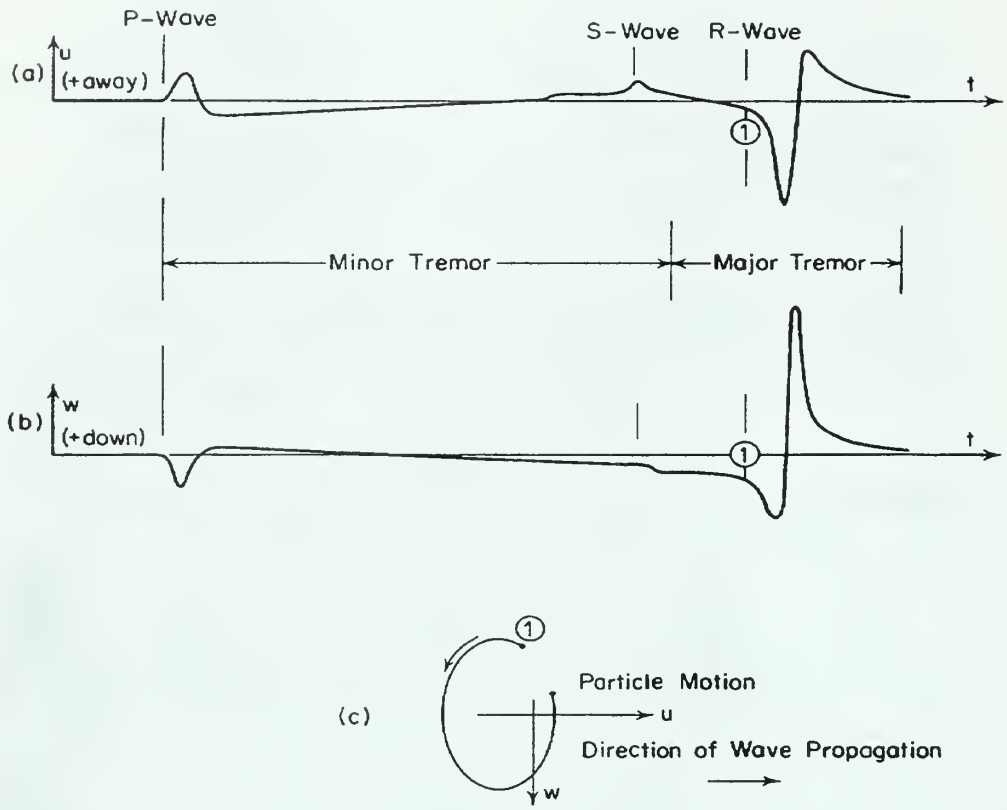
Source #2

Orientation: Vertical
Voltage: 10V (20Vpk-pk)
Frequency: 80Hz
Cycles: 1
Geophone #2
Filter: High Pass 40Hz
Gain: 40db

Comments: Overcast, med-high tide, 4-5ft waves.

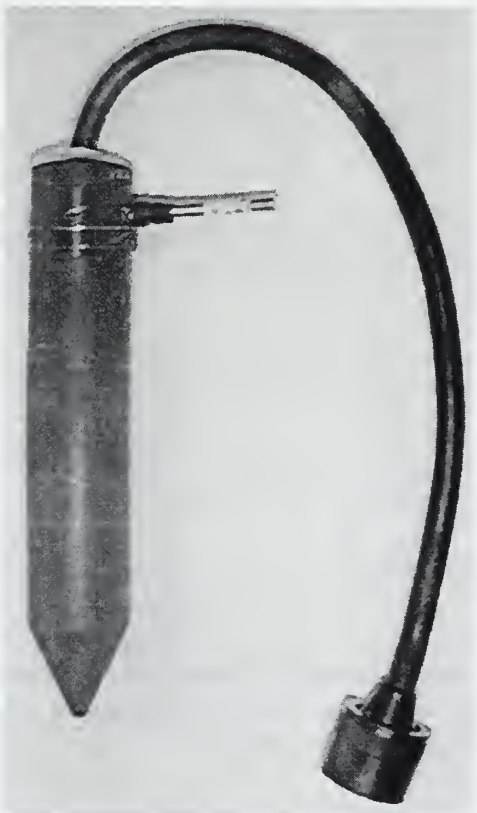


APPENDIX B7. Rayleigh wave [11]



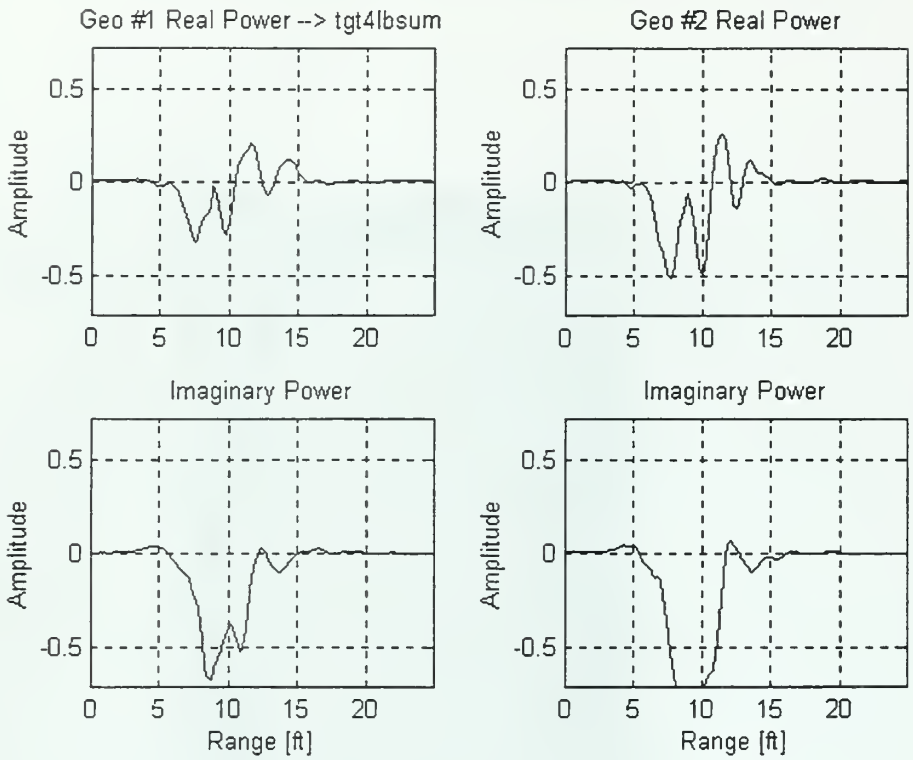
Seismic wavetrain resulting from a single vertical impulse source [11].

APPENDIX B8. GEOPHONE [11]



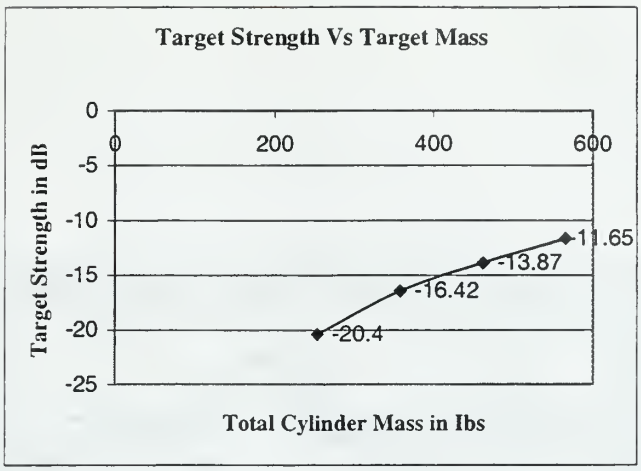
Triaxial geophone seismic sensor [11].

APPENDIX B9. CROSS POWER [12]

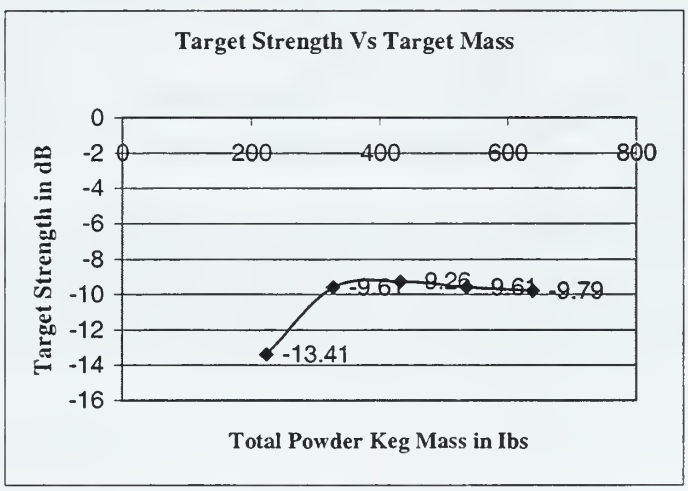


Real and imaginary cross power components of the received signal for both geophones. Target located at 22 feet.

APPENDIX B10. TARGETS STRENTH

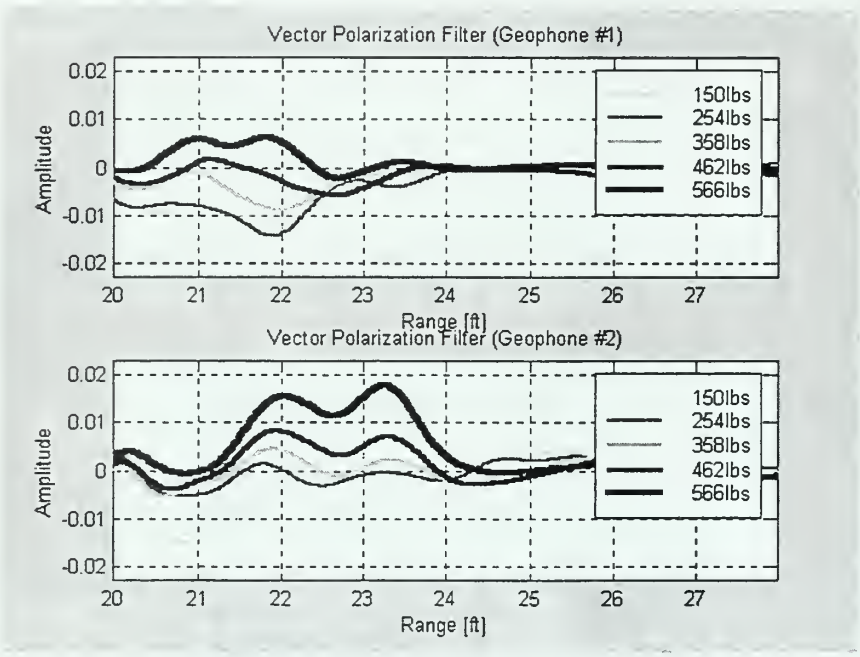


Target strength vs. target mass for cylinder target [12]



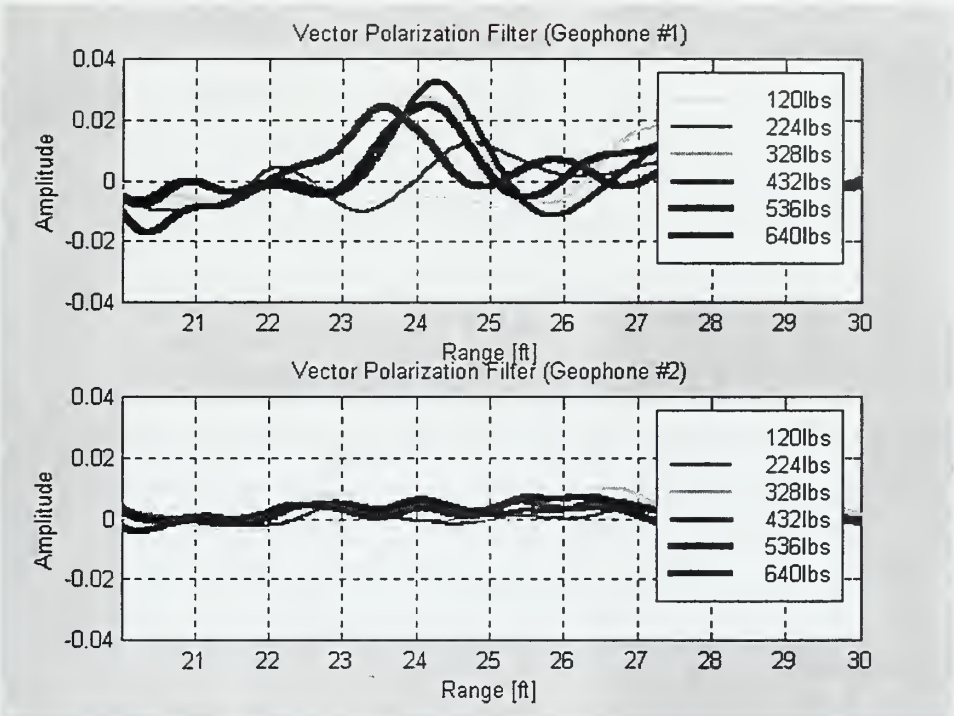
Target strength vs. target mass for powder keg target [12].

APPENDIX B10. IMAGINARY CROSS POWER SIGNAL FOR THE CYLINDER TARGET



Imaginary power plot for cylinder, November 6th experiment, 5 weight types [12].

APPENDIX B11. IMAGINARY CROSS POWER SIGNAL FOR THE POWDER KEG TARGET



Imaginary power plot for powder keg, November 10th experiment, six data plots. Note that the 2nd geophone did not receive the target [12].

APPENDIX C. MATLAB CODE; HMM BASED CLASSIFIER FOR MINE RECOGNITION

This Appendix contains the various MATLAB files used for recognizing the mine-like objects using HMMs. MATLAB files written in [11] used for preprocessing the data are also included for completeness.

APPENDIX C1. COMPUTATION OF THE CROSS POWER SIGNAL FROM RAW SIGNAL; 8 CHANNEL; SIGNAL. CODE WRITTEN BY M. FITZPATRICK [12]

```
% Name:         Present3.m
% Author:       LT Mike Fitzpatrick
% Updated:      9/31/98
% Description: This program conducts a Hilbert analysis....

%%%Input Parameters%%%
Range=1;                %Turns-on plotting with range axis
wavespd=295;            %Wavespeed [ft/s]
t_start=0.050;          %Set start time [s]
t_stop=0.150;           %Set stop time [s]
r_start=18;             %Set start range [ft]
r_stop=28;              %Set stop range [ft]
scale=0.0230;           %Set axis scaling (Set to 0 turns-off scaling)
georange=10;            %Enter range to geophone [ft]

%%%Date, Directory, & File%%%
date='Nov_6';
directory='Target3';
cd (date),cd (directory)

COUNT=0;                %Set start count
while(1)
   clc,disp('***Hilbert Analysis Subroutine***'),dir *.mat;
   COUNT=COUNT+1
   if COUNT==1,load tgt01bsum, end
   if COUNT==2,load tgt41bsum, end
   if COUNT==3,load tgt81bsum, end
   if COUNT==4,load tgt121bsum, end
   if COUNT==5,load tgt161bsum, end
   if COUNT==6,break,end
   [M,N]=size(channel);
   transform=hilbert(channel);
   Pwr(:,1,COUNT)=conj(transform(:,3)).*transform(:,5);
   Pwr(:,2,COUNT)=conj(transform(:,6)).*transform(:,8);
end
end

%%%Set Axes%%%
if Range==1
   [max1,index1]=max(abs(channel(:,1)));
   [max2,index2]=max(abs(channel(:,2)));
   index=round((index1+index2)/2);
   start=round(((index1+index2)/2)+(r_start/(dt*wavespd)));
   stop=round(((index1+index2)/2)+(r_stop/(dt*wavespd)));
   range=wavespd*(t(start:stop)-t(index));
   %start=round(delay+(r_start/(dt*wavespd)));
   %stop=round(delay+(r_stop/(dt*wavespd)));
   %range=wavespd*(t(start:stop)-t(delay));
else
   start=round((t_start/dt)+1);
   stop=round((t_stop/dt)+1);
```

```
end

if scale==0
   Realmax=1.1*max(max(abs(real(Pwr(start:stop,:)))));
   Imagmax=1.1*max(max(abs(imag(Pwr(start:stop,:)))));
else
   Realmax=scale; Imagmax=scale;
end

for n=1:COUNT-1
   Pwr1(:,n)=imag(Pwr(start:stop,1,n));
   Pwr2(:,n)=imag(Pwr(start:stop,2,n));
end

%%%Plotting%%%
figure,orient portrait
if Range==1
   subplot(2,1,1)
   plot(range,Pwr1(:,1),'c','LineWidth',1),hold
   plot(range,Pwr1(:,2),'m','LineWidth',2)
   plot(range,Pwr1(:,3),'g','LineWidth',3)
   plot(range,Pwr1(:,4),'b','LineWidth',4)
   plot(range,Pwr1(:,5),'r','LineWidth',5)
   axis([min(range) max(range) -Imagmax Imagmax]),hold
   title('Vector Polarization Filter (Geophone #1)')
   xlabel('Range [ft]'),ylabel('Amplitude'),grid
   legend('156lbs','160lbs','364lbs','468lbs','572lbs')
   %%%%%%%%%%%%
   subplot(2,1,2)
   plot(range,Pwr2(:,1),'c','LineWidth',1),hold
   plot(range,Pwr2(:,2),'m','LineWidth',2)
   plot(range,Pwr2(:,3),'g','LineWidth',3)
   plot(range,Pwr2(:,4),'b','LineWidth',4)
   plot(range,Pwr2(:,5),'r','LineWidth',5)
   axis([min(range) max(range) -Imagmax Imagmax]),hold
   title('Vector Polarization Filter (Geophone #2)')
   xlabel('Range [ft]'),ylabel('Amplitude'),grid
   legend('156lbs','160lbs','364lbs','468lbs','572lbs')
end
cd ..
cd ..
```

APPENDIX C2. DATA SELECTION USED FOR THE MULTIPLE WEIGHT SIGNAL EXPERIMENT SET-UP. DATA USED FOR HMM TRAINING

```
% Filename: kegcyl.m
% Written by: M. Zambartas
% Date Last Modified 10 August 1999
% Purpose: Generates and plots the signals for the multiple weight HMMs
training
%  kegmat.mat:  imaginary cross power of the keg signal
%  cylmat.mat:  imaginary cross power of the cylinder signal
%  back.mat:    imaginary cross power of a non target signal

load kegmat
load cylmat
load back;
range=linspace(1,60,3000);
% interpolation to 3000 points of all vectors, in order all signals to
have the same
% length

keg432lb(2277,:)=[];keg536lb(2277,:)=[];cyl572lb(2277,:)=[];
sk=size(keg224lb);sc=size(cyl364lb);sb=size(back);
kegint=zeros(3000,6);cyl1int=zeros(3000,6);backint=zeros(3000,6);
% 6 trials per signal:
for n=1:6

keg224lbint(:,n)=(interp1(1:sk(1),keg224lb(:,n),1:sk(1)/3001:sk(1)))';

keg432lbint(:,n)=(interp1(1:sk(1),keg432lb(:,n),1:sk(1)/3001:sk(1)))';

keg536lbint(:,n)=(interp1(1:sk(1),keg536lb(:,n),1:sk(1)/3001:sk(1)))';


cyl364lbint(:,n)=(interp1(1:sc(1),cyl364lb(:,n),1:sc(1)/3001:sc(1)))';

cyl468lbint(:,n)=(interp1(1:sc(1),cyl468lb(:,n),1:sc(1)/3001:sc(1)))';

cyl572lbint(:,n)=(interp1(1:sc(1),cyl572lb(:,n),1:sc(1)/3001:sc(1)))';

   backint(:,n)=(interp1(1:sb(1),back(:,n),1:sb(1)/3001:sb(1)))';
end

% plot of all signals
subplot(3,2,1)
   plot(range,keg224lbint(:,1),'c','LineWidth',1),hold
   plot(range,keg224lbint(:,2),'m','LineWidth',1)
   plot(range,keg224lbint(:,3),'g','LineWidth',1)
   plot(range,keg224lbint(:,4),'b','LineWidth',1)
   plot(range,keg224lbint(:,5),'k','LineWidth',1)
   plot(range,keg224lbint(:,6),'r','LineWidth',1)
   hold
   title('Powder Keg 224lb  ')

axis([0 60 -0.03 0.03]),grid;
subplot(3,2,3)
```

```
   plot(range,keg432lbint(:,1),'c','LineWidth',1),hold
   plot(range,keg432lbint(:,2),'m','LineWidth',1)
   plot(range,keg432lbint(:,3),'g','LineWidth',1)
   plot(range,keg432lbint(:,4),'b','LineWidth',1)
   plot(range,keg432lbint(:,5),'k','LineWidth',1)
   plot(range,keg432lbint(:,6),'r','LineWidth',1)
   axis([0 60 -0.03 0.03]),grid;
   title('Powder Keg 432lb ')

hold
subplot(3,2,5)
   plot(range,keg536lbint(:,1),'c','LineWidth',1),hold
   plot(range,keg536lbint(:,2),'m','LineWidth',1)
   plot(range,keg536lbint(:,3),'g','LineWidth',1)
   plot(range,keg536lbint(:,4),'b','LineWidth',1)
   plot(range,keg536lbint(:,5),'k','LineWidth',1)
   plot(range,keg536lbint(:,6),'r','LineWidth',1)
   axis([0 60 -0.03 0.03]),grid;
   title('Powder Keg 536lb ')

hold
subplot(3,2,2)
   plot(range,cyl364lbint(:,1),'c','LineWidth',1),hold
   plot(range,cyl364lbint(:,2),'m','LineWidth',1)
   plot(range,cyl364lbint(:,3),'g','LineWidth',1)
   plot(range,cyl364lbint(:,4),'b','LineWidth',1)
   plot(range,cyl364lbint(:,5),'k','LineWidth',1)
   plot(range,cyl364lbint(:,6),'r','LineWidth',1)
axis([0 60 -0.02 0.02]),grid;
   title('Cylinder 364lb ')
hold
  subplot(3,2,4)
   plot(range,cyl468lbint(:,1),'c','LineWidth',1),hold
   plot(range,cyl468lbint(:,2),'m','LineWidth',1)
   plot(range,cyl468lbint(:,3),'g','LineWidth',1)
   plot(range,cyl468lbint(:,4),'b','LineWidth',1)
   plot(range,cyl468lbint(:,5),'k','LineWidth',1)
   plot(range,cyl468lbint(:,6),'r','LineWidth',1)
axis([0 60 -0.02 0.02]),grid;
   title('Cylinder 468lb ')
hold
 subplot(3,2,6)
   plot(range,cyl572lbint(:,1),'c','LineWidth',1),hold
   plot(range,cyl572lbint(:,2),'m','LineWidth',1)
   plot(range,cyl572lbint(:,3),'g','LineWidth',1)
   plot(range,cyl572lbint(:,4),'b','LineWidth',1)
   plot(range,cyl572lbint(:,5),'k','LineWidth',1)
   plot(range,cyl572lbint(:,6),'r','LineWidth',1)
axis([0 60 -0.02 0.02]),grid;
   title('Cylinder 572lb ')
hold
% target location:
for n=1:6
   ik224(n)=find
(keg224lbint(1000:1800,n)==max(keg224lbint(1000:1800,n)));
   ik432(n)=find
(keg432lbint(1000:1800,n)==max(keg432lbint(1000:1800,n)));
```

```
   ik536(n)=find
(keg536lbint(1000:1800,n)==max(keg536lbint(1000:1800,n)));
   ik224(n)=ik224(n)+1000-100;
   ik432(n)=ik432(n)+1000-100;
   ik536(n)=ik536(n)+1000-100;
end
ik224=ik224(1);ik432=ik432(1);ik536=ik536(1);
for n=1:6
   ic364(n)=find
(cyl364lbint(1000:1800,n)==max(cyl364lbint(1000:1800,n)));
   ic364(n)=ic364(n)+1000-100;
   ic468(n)=find
(cyl468lbint(1000:1800,n)==max(cyl468lbint(1000:1800,n)));
   ic468(n)=ic468(n)+1000-100;
ic572(n)=find
(cyl572lbint(1000:1800,n)==max(cyl572lbint(1000:1800,n)));
   ic572(n)=ic572(n)+1000-100;


end
ic364=ic364(1);ic468=ic468(1);ic572=ic572(6);

% so now we know where the target is located, we can build the signal
file
% signal(data,mine,trial)
% mine(1-3) is for the keg (6,8,10 ft)' s target, mine(4) is for the
cylinder' s target
% and mine(5-10) background and other non target data
signalmulti=zeros(400,10,6);
for trials=1:6
   signalmulti(:,1,trials)=keg224lbint(ik224:ik224+399,trials);
   signalmulti(:,2,trials)=keg432lbint(ik432:ik432+399,trials);
   signalmulti(:,3,trials)=keg536lbint(ik536:ik536+399,trials);
   signalmulti(:,4,trials)=cyl364lbint(ic364:ic364+399,trials);
      signalmulti(:,5,trials)=cyl468lbint(ic468:ic468+399,trials);
   signalmulti(:,6,trials)=cyl572lbint(ic572:ic572+399,trials);

   signalmulti(:,7,trials)=backint(ic364:ic364+399,trials);
   signalmulti(:,8,trials)=backint(ik536:ik536+399,trials);
   signalmulti(:,9,trials)=backint(2001:2400,trials);
   signalmulti(:,10,trials)=backint(1500:1899,trials);
   signalmulti(:,11,trials)=backint(800:800+399,trials);
   signalmulti(:,12,trials)=backint(2500:2899,trials);

end
%signalmulti(:,3,:)=signalmulti(:,3,:)/2;
for m=1:4
for n=1:6
   signalmulti(:,m,n)=signalmulti(:,m,n)-mean(signalmulti(:,m,n));
end
end
cd ..
save signalmulti signalmulti
cd data
figure(2)
range=1:400;
%range=linspace(21,29,400);
```

```
subplot(3,2,1)
   plot(range,signalmulti(:,1,1),'c','LineWidth',1),hold
   plot(range,signalmulti(:,1,2),'m','LineWidth',1)
   plot(range,signalmulti(:,1,3),'g','LineWidth',1)
   plot(range,signalmulti(:,1,4),'b','LineWidth',1)
   plot(range,signalmulti(:,1,5),'k','LineWidth',1)
   plot(range,signalmulti(:,1,6),'r','LineWidth',1)
   title('Powder Keg 224lb  ')
   hold
subplot(3,2,3)
   plot(range,signalmulti(:,2,1),'c','LineWidth',1),hold
   plot(range,signalmulti(:,2,2),'m','LineWidth',1)
   plot(range,signalmulti(:,2,3),'g','LineWidth',1)
   plot(range,signalmulti(:,2,4),'b','LineWidth',1)
   plot(range,signalmulti(:,2,5),'k','LineWidth',1)
   plot(range,signalmulti(:,2,6),'r','LineWidth',1)
   title('Powder Keg 432lb ')
   hold
subplot(3,2,5)
   plot(range,signalmulti(:,3,1),'c','LineWidth',1),hold
   plot(range,signalmulti(:,3,2),'m','LineWidth',1)
   plot(range,signalmulti(:,3,3),'g','LineWidth',1)
   plot(range,signalmulti(:,3,4),'b','LineWidth',1)
   plot(range,signalmulti(:,3,5),'k','LineWidth',1)
   plot(range,signalmulti(:,3,6),'r','LineWidth',1)
   title('Powder Keg 536lb ')
     %axis([0 400 -0.02 0.02])
     xlabel ('Data Points')
   hold
subplot(3,2,2)
   plot(range,signalmulti(:,4,1),'c','LineWidth',1),hold
   plot(range,signalmulti(:,4,2),'m','LineWidth',1)
   plot(range,signalmulti(:,4,3),'g','LineWidth',1)
   plot(range,signalmulti(:,4,4),'b','LineWidth',1)
   plot(range,signalmulti(:,4,5),'k','LineWidth',1)
   plot(range,signalmulti(:,4,6),'r','LineWidth',1)
   title('Cylinder 364lb ')
   hold
   subplot(3,2,4)
   plot(range,signalmulti(:,5,1),'c','LineWidth',1),hold
   plot(range,signalmulti(:,5,2),'m','LineWidth',1)
   plot(range,signalmulti(:,5,3),'g','LineWidth',1)
   plot(range,signalmulti(:,5,4),'b','LineWidth',1)
   plot(range,signalmulti(:,5,5),'k','LineWidth',1)
   plot(range,signalmulti(:,5,6),'r','LineWidth',1)

   title('Cylinder 468lb ')

   hold
subplot(3,2,6)
   plot(range,signalmulti(:,6,1),'c','LineWidth',1),hold
   plot(range,signalmulti(:,6,2),'m','LineWidth',1)
   plot(range,signalmulti(:,6,3),'g','LineWidth',1)
   plot(range,signalmulti(:,6,4),'b','LineWidth',1)
   plot(range,signalmulti(:,6,5),'k','LineWidth',1)
   plot(range,signalmulti(:,6,6),'r','LineWidth',1)
   title('Cylinder 572lb ')
```

```
xlabel ('Data Points')
hold
cd ..

save signal signalmulti
cd dat
```

APPENDIX C3. FEATURE ANALYSIS FOR MINE LIKE OBJECT SIGNALS; MULTIPLE WEIGHTS EXPERIMENT SET-UP

```
% Filename: trainingkegcyl.m
% Written by: M. Zambartas
% Date Last Modified 10 August 1999

% Purpose: Features Analysis  for mine-like object signals with
multiple weights
%          (3 sets of weights for the keg and the cylinder - 6 trials
each, and 7 background signals), for
%          HMM recognition use.  We will create two classes, one for
every mine-like object.
%          Vector quantization using k-means
% M: # of symbols
% N: # of states
% mines: # total # of mines
% c:     # coefficients matrix for all mines
% w:     # VQ coeff matrix




% vector quantization  with multiple lb (keg,cylinder,background)
 clear
load signal
signal=signalmulti;
clear signalmulti;
M=8;N=4;
rand('seed',1);
n_it=20;
%segs=4;
mines=12;


s=size(signal);s1=s(1);


%seg=fix(s1/segs);

%Coefficients evaluation
%c=zeros(T,4,8,segs,6);
for mine=1:mines
for trial=1:6

      [c(:,:,mine,trial),T]=coeffinterp(signal(:,mine,trial));

end
  end



% vector quantization
p1=[];
```

```
for mine=1:mines
for trial=1:6
  p1=[p1 ;c(:,:,mine,trial)];
end
end
[CODE, label, dist] = svq(p1, M, n_it);

class=zeros(T,mines,5);
w=zeros(T,8,mines,5);
for mine=1:mines
 for trial=1:6
   [w class(:,mine,trial), dist]=vq(c(:,:,mine,trial),CODE,n_it);
 end
end

% Test of vector quantization
%figure(1)
%subplot(4,2,1);plot(1:8,c(1,:,1,1),'*',1:8,w(class(1,1,1),:,1,1)),titl
e(['vector1,Class=' num2str(class(1,1,1)) ' M=' num2str(M)])
%subplot(4,2,2);plot(1:8,c(1,:,1,2),'*',1:8,w(class(1,1,2),:,1,2)),titl
e(['vector2,Class=' num2str(class(1,1,1)) ' M=' num2str(M)])
%qa(1:T,:)=w(classmicro(1:T),:);
%qa(1:T,:)=w(classsta(1:T),:);
distances=[];
%for n=1:segs
 %  for trial=1:5
 %  distances= [distances
dist(c(:,:,n,trial),w(class(:,n,trial),:,n,trial)')];
%end


%end
%figure(3)
%stem(distances),title('Euclidean Distance original/quantized vectors')
```

APPENDIX C4. SIGNAL DECIMATION

```
% Filename: coeffinterp.m
% Written by: M. Zambartas
% Date Last Modified 10 August 1999
% Purpose: fuction that performs a decimation of segment and normalizes
it.
%  T:  total number of segments
%  c:  Coefficients vector

function [c,T]=coeffinterp(word);
word=word-mean(word);

%int=interp1(1:l,word,1:l/10000:10000);

%pre=filter([1,-.98],1,word);
pre=word;
% hanning filter:
%han=hanning(3);
%pre=conv(han,pre);
l=length(word);

%start=1;
step=l/2;
%l=2000;
% 0% overlap
overlap=round((0/100)*step);
%# of Observations T:
T=l/step+(l*overlap/step^2);
T=floor(T);
for n=1:T
 c(n,1:10)=interp1(1:step,pre(1+step*(n-1)-overlap*(n-1):step*n-
overlap*(n-1)),1:step/10:step);
 end
%c(:,1)=c(:,1)/max(c(:,1));
%c(:,1)=c(:,1)-.5;
% normalization:
for n=1:T
   c(n,:)=c(n,:)./max(abs(c(n,:)));
end
```

APPENDIX C5. HMM TRAINING AND SCORING FOR MULTIPLE WEIGHTS MINE LIKE SIGNAL DATA

```
% Filename: sequence.m
% Written by: M. Zambartas
% Date Last Modified 10 August 1999
% Purpose: Performs all the sequence of HMM training and scoring the
signals from multiple lbs mines
%          rotating every time the tested signalkegmat.mat:  imaginary
cross power of the keg signal

%  cylmat.mat:  imaginary cross power of the cylinder signal
%  back.mat:    imaginary cross power of a non target signal

% testing sequence
seq=[2 3 4 5 6
     1 3 4 5 6
     1 2 4 5 6
     1 2 3 5 6
     1 2 3 4 6
     1 2 3 4 5];
for test=1:6;
   ts=seq(test,:);
   % newmodelkegcyl trains the HMM using the seq sequence of signals
   newmodelkegcyl
   % scretestkegcyl scores the HMM for every testing signals
   scoretestkegcyl
   P_bkeg224(test)=P_bwk224;
   P_vkeg224(test)=P_vk224;
   P_bkeg432(test)=P_bwk432;
   P_vkeg432(test)=P_vk432;
   P_bkeg536(test)=P_bwk536;
   P_vkeg536(test)=P_vk536;
   P_bcyl364(test)=P_bwc364;
   P_vcyl364(test)=P_vc364;
   P_bcyl468(test)=P_bwc468;
   P_vcyl468(test)=P_vc468;
   P_bcyl572(test)=P_bwc572;
   P_vcyl572(test)=P_vc572;
   P_bback(test)=P_bwb;
   P_vback(test)=P_vb;

end
```

APPENDIX C6. GENERATION OF HMM OBSERVATION SEQUENCE FOR MULTIPLE WEIGHTS MINE LIKE SIGNAL DATA

```
% Filename: newmodelkegcyl.m
% Written by: M. Zambartas
% Date Last Modified 10 August 1999
% Purpose: Creates the multiple observation matrix O_multi for multiple
observations
%           HMM training of multi lbs testing
% ts: traininf sequence, assigned at sequence.m file
% training sequence:

% keg (224,432,536lb):
%O_multi=[class(:,1,ts(1))';class(:,1,ts(2))';class(:,1,ts(3))';class(:
,1,ts(4))';class(:,1,ts(5))';class(:,2,ts(1))';class(:,2,ts(2))';class(
:,2,ts(3))';class(:,2,ts(4))';class(:,2,ts(5))';class(:,3,ts(1))';class
(:,3,ts(2))';class(:,3,ts(3))';class(:,3,ts(4))';class(:,3,ts(5))'];
% cylinder (364,468,572lb)
O_multi=[class(:,4,ts(1))';class(:,4,ts(2))';class(:,4,ts(3))';class(:,
4,ts(4))';class(:,4,ts(5))';class(:,5,ts(1))';class(:,5,ts(2))';class(:
,5,ts(3))';class(:,5,ts(4))';class(:,5,ts(5))';class(:,6,ts(1))';class(
:,6,ts(2))';class(:,6,ts(3))';class(:,6,ts(4))';class(:,6,ts(5))'];

[a,b,pi] = trainmulti(O_multi,N,T,M);
```

APPENDIX C7. SCORING FOR THE MULTIPLE WEIGHT MINE LIKE SIGNAL SET-UP

```
% Filename:scorekegcyl.m
% Written by: M. Zambartas
% Date Last Modified 10 August 1999
% Purpose: Scoring of all testing mine-like signals-multiple lbs case


'test: keg224lb'

O=class(:,1,test);
[P_bwk224,P_vk224]=score(a,b,pi,O')
if (P_bwk224==0)
   P_bwk224=eps;
end
if (P_vk224==0)
   P_vk224=eps;
end
'test: keg432lb'
O=class(:,2,test);
[P_bwk432,P_vk432]=score(a,b,pi,O')
if (P_bwk432==0)
   P_bwk432=eps;
end
if (P_vk432==0)
   P_vk432=eps;
end
'test: keg536lb'
O=class(:,3,test);
[P_bwk536,P_vk536]=score(a,b,pi,O')
if (P_bwk536==0)
   P_bwk536=eps;
end
if (P_vk536==0)
   P_vk536=eps;
end

'test: cylinder364lb'
O=class(:,4,test);
[P_bwc364,P_vc364]=score(a,b,pi,O')
if (P_bwc364==0)
   P_bwc364=eps;
end
if (P_vc364==0)
   P_vc364=eps;
end

'test: cylinder468lb'
O=class(:,5,test);
[P_bwc468,P_vc468]=score(a,b,pi,O')
if (P_bwc468==0)
   P_bwc468=eps;
end
if (P_vc468==0)
```

```
   P_vc468=eps;
end

'test: cylinder572lb'
O=class(:,6,test);
[P_bwc572,P_vc572]=score(a,b,pi,O')
if (P_bwc572==0)
   P_bwc572=eps;
end
if (P_vc572==0)
   P_vc572=eps;
end

'test: background'
O=class(:,11,test);
[P_bwb,P_vb]=score(a,b,pi,O')
if (P_bwb==0)
   P_bwb=eps;
end
if (P_vb==0)
   P_vb=eps;
end



%figure
subplot(6,2,test*2-1), barh(10*log10([P_bwk224 P_bwk432 P_bwk536
P_bwc364 P_bwc468 P_bwc572 P_bwb]));
if test==1
title(['                  Pr[O|lamda(cylinder)] - 1:keg_2_2_4_1_b,
2:keg_4_3_2_1_b, 3:keg_5_3_6_1_b, 4:cylinder_3_6_4_1_b,
5:cylinder_4_6_8_1_b, 6:cylinder_5_7_2_1_b, 7:background, M='
num2str(M) ', N=' num2str(N)])
end
axis([-50 0 1 7]);
if test==6
   xlabel('P_V_i_t_e_r_b_i[dB]')
end
if test==6
   xlabel('P_B_W[dB]')
end
if test==3
ylabel('tested segment')
end

if test==3
   ylabel('tested segment')
end

grid
hold
subplot(6,2,test*2), barh(10*log10([P_vk224 P_vk432 P_vk536 P_vc364
P_vc468 P_vc572 P_vb]));
%title(['Viterbi Probability  - 1:keg_224lb, 2:keg_432lb, 3:keg_536lb,
4:cylinder_364lb, 5:cylinder_468lb, 6:cylinder_572lb, 7:background, M='
num2str(M) ', N=' num2str(N)]), xlabel('P_V_i_t_e_r_b_i[dB]'),
ylabel('tested segment')
```

```
axis([-50 0 1 7]);
grid
if test==6
   xlabel('P_V_i_t_e_r_b_i[dB]')
end
if test==3
   ylabel('tested segment')
end
hold
```

APPENDIX C8. DATA SELECTION FOR THE MULTIPLE DISTANCE SIGNAL EXPERIMENT SETUP. DATA USED FOR HMM TRAINING

```
% Filename: multift.m
% Written by: M. Zambartas
% Date Last Modified 10 August 1999
% Purpose: Generates and plots the signals for the multiple FT HMMs
training
%  kegmat.mat:  imaginary cross power of the keg signal
%  cylmat.mat:  imaginary cross power of the cylinder signal
%  back.mat:    imaginary cross power of a non target signal


load multiftkeg
load cyl6noe
load back;
range=linspace(1,60,3000);
keg6ft(2276,:)=[];keg10ft(2276,:)=[];
sk=size(keg6ft);sc=size(cyl6noe);sb=size(back);
kegint=zeros(3000,6);cyl1int=zeros(3000,6);backint=zeros(3000,6);

% interpolation to 3000 points
for n=1:6
   keg6ftint(:,n)=(interp1(1:sk(1),keg6ft(:,n),1:sk(1)/3001:sk(1)))';
   keg8ftint(:,n)=(interp1(1:sk(1),keg8ft(:,n),1:sk(1)/3001:sk(1)))';
   keg10ftint(:,n)=(interp1(1:sk(1),keg10ft(:,n),1:sk(1)/3001:sk(1)))';

   cyl6noeint(:,n)=(interp1(1:sc(1),cyl6noe(:,n),1:sc(1)/3001:sc(1)))';
   backint(:,n)=(interp1(1:sb(1),back(:,n),1:sb(1)/3001:sb(1)))';
end


subplot(4,1,1)
   plot(range,keg6ftint(:,1),'c','LineWidth',1),hold
   plot(range,keg6ftint(:,2),'m','LineWidth',1)
   plot(range,keg6ftint(:,3),'g','LineWidth',1)
   plot(range,keg6ftint(:,4),'b','LineWidth',1)
   plot(range,keg6ftint(:,5),'k','LineWidth',1)
   plot(range,keg6ftint(:,6),'r','LineWidth',1)
   hold
   title('Powder Keg 6ft  (total:24ft)')

axis([0 60 -0.02 0.02]),grid;
subplot(4,1,2)
   plot(range,keg8ftint(:,1),'c','LineWidth',1),hold
   plot(range,keg8ftint(:,2),'m','LineWidth',1)
   plot(range,keg8ftint(:,3),'g','LineWidth',1)
   plot(range,keg8ftint(:,4),'b','LineWidth',1)
   plot(range,keg8ftint(:,5),'k','LineWidth',1)
   plot(range,keg8ftint(:,6),'r','LineWidth',1)
   axis([0 60 -0.02 0.02]),grid;
   title('Powder Keg 8ft  (total:22ft)')

hold
subplot(4,1,3)
```

```
   plot(range,keg10ftint(:,1),'c','LineWidth',1),hold
   plot(range,keg10ftint(:,2),'m','LineWidth',1)
   plot(range,keg10ftint(:,3),'g','LineWidth',1)
   plot(range,keg10ftint(:,4),'b','LineWidth',1)
   plot(range,keg10ftint(:,5),'k','LineWidth',1)
   plot(range,keg10ftint(:,6),'r','LineWidth',1)
   axis([0 60 -0.02 0.02]),grid;
   title('Powder Keg 10ft  (total:20ft)')

hold
range=1:3000;
subplot(4,1,4)
   plot(range,cyl6noeint(:,1),'c','LineWidth',1),hold
   plot(range,cyl6noeint(:,2),'m','LineWidth',1)
   plot(range,cyl6noeint(:,3),'g','LineWidth',1)
   plot(range,cyl6noeint(:,4),'b','LineWidth',1)
   plot(range,cyl6noeint(:,5),'k','LineWidth',1)
   plot(range,cyl6noeint(:,6),'r','LineWidth',1)
axis([0 3000 -0.02 0.02]),grid;
   title('Cylinder 10ft  (total:20ft)')
hold
for n=1:6
   ik6(n)=find (keg6ftint(1000:1800,n)==max(keg6ftint(1000:1800,n)));
   ik8(n)=find (keg8ftint(1000:1800,n)==max(keg8ftint(1000:1800,n)));
   ik10(n)=find
(keg10ftint(1000:1800,n)==max(keg10ftint(1000:1800,n)));
   ik6(n)=ik6(n)+1000-100;
   ik8(n)=ik8(n)+1000-100;
   ik10(n)=ik10(n)+1000-100;
end
ik6=ik6(1);ik8=ik8(1);ik10=ik10(1);
for n=1:6
   ic10(n)=find
(cyl6noeint(1000:1800,n)==max(cyl6noeint(1000:1800,n)));
   ic10(n)=ic10(n)+1000-100;
end
ic10=ic10(1)-100;
% so now we know where the target is located, we can build the signal
file
% signal(data,mine,trial)
% mine(1-3) is for the keg (6,8,10 ft)' s target, mine(4) is for the
cylinder' s target
% and mine(5-10) background and other non target data
signalmulti=zeros(400,10,6);
for trials=1:6
   signalmulti(:,1,trials)=keg6ftint(ik6:ik6+399,trials);
   signalmulti(:,2,trials)=keg8ftint(ik8:ik8+399,trials);
   signalmulti(:,3,trials)=keg10ftint(ik10:ik10+399,trials);
   signalmulti(:,4,trials)=cyl6noeint(ic10:ic10+399,trials);
   signalmulti(:,5,trials)=backint(ic10:ic10+399,trials);
   signalmulti(:,6,trials)=backint(ik10:ik10+399,trials);
   signalmulti(:,7,trials)=backint(2001:2400,trials);
   signalmulti(:,8,trials)=backint(1500:1899,trials);
   signalmulti(:,9,trials)=backint(800:800+399,trials);
   signalmulti(:,10,trials)=backint(2500:2899,trials);

end
```

```
signalmulti(:,3,:)=signalmulti(:,3,:)/3;
for m=1:4
for n=1:6
   signalmulti(:,m,n)=signalmulti(:,m,n)-mean(signalmulti(:,m,n));
end
end
cd ..
save signalmulti signalmulti
cd data
figure(2)
range=1:400;
subplot(4,1,1)
   plot(range,signalmulti(:,1,1),'c','LineWidth',1),hold
   plot(range,signalmulti(:,1,2),'m','LineWidth',1)
   plot(range,signalmulti(:,1,3),'g','LineWidth',1)
   plot(range,signalmulti(:,1,4),'b','LineWidth',1)
   plot(range,signalmulti(:,1,5),'k','LineWidth',1)
   plot(range,signalmulti(:,1,6),'r','LineWidth',1)
   hold
subplot(4,1,2)
   plot(range,signalmulti(:,2,1),'c','LineWidth',1),hold
   plot(range,signalmulti(:,2,2),'m','LineWidth',1)
   plot(range,signalmulti(:,2,3),'g','LineWidth',1)
   plot(range,signalmulti(:,2,4),'b','LineWidth',1)
   plot(range,signalmulti(:,2,5),'k','LineWidth',1)
   plot(range,signalmulti(:,2,6),'r','LineWidth',1)
   hold
subplot(4,1,3)
   plot(range,signalmulti(:,3,1),'c','LineWidth',1),hold
   plot(range,signalmulti(:,3,2),'m','LineWidth',1)
   plot(range,signalmulti(:,3,3),'g','LineWidth',1)
   plot(range,signalmulti(:,3,4),'b','LineWidth',1)
   plot(range,signalmulti(:,3,5),'k','LineWidth',1)
   plot(range,signalmulti(:,3,6),'r','LineWidth',1)
   hold
subplot(4,1,4)
   plot(range,signalmulti(:,4,1),'c','LineWidth',1),hold
   plot(range,signalmulti(:,4,2),'m','LineWidth',1)
   plot(range,signalmulti(:,4,3),'g','LineWidth',1)
   plot(range,signalmulti(:,4,4),'b','LineWidth',1)
   plot(range,signalmulti(:,4,5),'k','LineWidth',1)
   plot(range,signalmulti(:,4,6),'r','LineWidth',1)
   hold
```

APPENDIX C9. FEATURE ANALYSIS FOR MINE LIKE OBJECT SIGNALS; MULTIPLE DISTANCES EXPERIMENT SET-UP

```
% Filename: trainingkmmulti.m
% Written by: M. Zambartas
% Date Last Modified 10 August 1999

% Purpose: Features Analysis  for mine-like object signals with
multiple ft
%          (3 sets of ft for the keg and  one for the cylinder - 6
trials each, and 6 trials of background signals), for
%          HMM recognition use.  We will create two classes, one for
every mine-like object.
%          Vector quantization using k-means
% M: # of symbols
% N: # of states
% mines: # total # of mines
% c:     # coefficients matrix for all mines
% w:     # VQ coeff matrix

 clear
load signalmulti
signal=signalmulti;
clear signalmulti;
M=8;N=4;
rand('seed',1);
n_it=20;
%segs=4;
mines=10;


s=size(signal);s1=s(1);


%seg=fix(s1/segs);

%Coefficients evaluation
%c=zeros(T,4,8,segs,6);
for mine=1:mines
for trial=1:6

      [c(:,:,mine,trial),T]=coeffinterp(signal(:,mine,trial));

end
  end



% vector quantization
p1=[];
for mine=1:mines
for trial=1:6
  p1=[p1 ;c(:,:,mine,trial)];
end
end
```

```
[CODE, label, dist] = svq(p1, M, n_it);

class=zeros(T,mines,5);
w=zeros(T,8,mines,5);
for mine=1:mines
 for trial=1:6
   [w class(:,mine,trial), dist]=vq(c(:,:,mine,trial),CODE,n_it);
 end
end

% Test of vector quantization
%figure(1)
%subplot(4,2,1);plot(1:8,c(1,:,1,1),'*',1:8,w(class(1,1,1),:,1,1)),titl
e(['vector1,Class=' num2str(class(1,1,1)) ' M=' num2str(M)])
%subplot(4,2,2);plot(1:8,c(1,:,1,2),'*',1:8,w(class(1,1,2),:,1,2)),titl
e(['vector2,Class=' num2str(class(1,1,1)) ' M=' num2str(M)])
%qa(1:T,:)=w(classmicro(1:T),:);
%qa(1:T,:)=w(classsta(1:T),:);
distances=[];
%for n=1:segs
 %  for trial=1:5
 %  distances= [distances
dist(c(:,:,n,trial),w(class(:,n,trial),:,n,trial)')];
%end


%end
%figure(3)
%stem(distances),title('Euclidean Distance original/quantized vectors')
```

APPENDIX C10. HMM TRAINING AND SCORING FOR MULTIPLE DISTANCE SIGNAL MINE LIKE SIGNAL DATA

```
% Filename: sequencemulti.m
% Written by: M. Zambartas
% Date Last Modified 10 August 1999
% Purpose: Performs all the sequence of HMM training and scoring the
signals from multiple ft mines
%          rotating every time the tested signalkegmat.mat:  imaginary
cross power of the keg signal
%  cylmat.mat:  imaginary cross power of the cylinder signal
%  back.mat:    imaginary cross power of a non target signal



% testing sequence for multi target distance
% rotation of the testing signal:
seq=[2 3 4 5 6
     1 3 4 5 6
     1 2 4 5 6
     1 2 3 5 6
     1 2 3 4 6
     1 2 3 4 5];
for test=1:6;
   ts=seq(test,:);
   newmodelmulti
   scoretestmulti
   P_bkeg6ft(test)=P_bwk6;
   P_vkeg6ft(test)=P_vk6;
   P_bkeg8ft(test)=P_bwk8;
   P_vkeg8ft(test)=P_vk8;
   P_bkeg10ft(test)=P_bwk10;
   P_vkeg10ft(test)=P_vk10;
   P_bcyl10ft(test)=P_bwc;
   P_vcyl10ft(test)=P_vc;
   P_bback(test)=P_bwb;
   P_vback(test)=P_vb;

end
```

APPENDIX C11. GENERATION OF HMM OBSERVATION SEQUENCE FOR MULTIPLE DISTANCES MINE LIKE SIGNAL DATA

```
% Filename: newmodelmulti.m
% Written by: M. Zambartas
% Date Last Modified 10 August 1999
% Purpose: Creates the multiple observation matrix O_multi for multiple
observation
%           HMM training of multi ft testing
% ts: traininf sequence, assigned at sequence.m file
% training sequence:


% initial conditions:
% class(coefficient(1-T),segment(1-4),mine,mass)
% keg (6,8,10ft):
O_multi=[class(:,1,ts(1))';class(:,1,ts(2))';class(:,1,ts(3))';class(:,
1,ts(4))';class(:,1,ts(5))';class(:,2,ts(1))';class(:,2,ts(2))';class(:
,2,ts(3))';class(:,2,ts(4))';class(:,2,ts(5))';class(:,3,ts(1))';class(
:,3,ts(2))';class(:,3,ts(3))';class(:,3,ts(4))';class(:,3,ts(5))'];
% cylinder (single 10 ft)
%O_multi=[class(:,4,ts(1))';class(:,4,ts(2))';class(:,4,ts(3))';class(:
,4,ts(4))';class(:,4,ts(5))'];

 [a,b,pi] = trainmulti(O_multi,N,T,M);
```

APPENDIX C12. TESTING DATA; SCORING FOR THE MULTIPLE DISTANCES MINE LIKE SIGNAL SET-UP

```
% Filename: scoretestmulti.m
% Written by: M. Zambartas
% Date Last Modified 10 August 1999
% Purpose: Scoring of all testing mine-like signals-multiple ft case

'test: keg6ft'
O=class(:,1,test);
[P_bwk6,P_vk6]=score(a,b,pi,O')
if (P_bwk6==0)
   P_bwk6=eps;
end
if (P_vk6==0)
   P_vk6=eps;
end
'test: keg8ft'
O=class(:,2,test);
[P_bwk8,P_vk8]=score(a,b,pi,O')
if (P_bwk8==0)
   P_bwk8=eps;
end
if (P_vk8==0)
   P_vk8=eps;
end
'test: keg10ft'
O=class(:,3,test);
[P_bwk10,P_vk10]=score(a,b,pi,O')
if (P_bwk10==0)
   P_bwk10=eps;
end
if (P_vk10==0)
   P_vk10=eps;
end

'test: cylinder'
O=class(:,4,test);
[P_bwc,P_vc]=score(a,b,pi,O')
if (P_bwc==0)
   P_bwc=eps;
end
if (P_vc==0)
   P_vc=eps;
end

'test: background'
O=class(:,8,test);
[P_bwb,P_vb]=score(a,b,pi,O')
if (P_bwb==0)
   P_bwb=eps;
end
if (P_vb==0)
   P_vb=eps;
end
```

```
subplot(6,2,test*2-1), barh(10*log10([P_bwk6 P_bwk8 P_bwk10 P_bwc
P_bwb]));
if test==1
   title(['                        Pr(O|lamda_k_e_g) - 1:keg_6ft,
2:keg_8ft, 3:keg_1_0ft, 4:cylinder, test:background, M=' num2str(M) ',
N=' num2str(N),', T=2 '])

end
if test==6
   xlabel('P_B_W[dB]')
end
if test==3
ylabel('tested signal')
end

axis([-50 0 1 5]);
grid
subplot(6,2,test*2), barh(10*log10([P_vk6 P_vk8 P_vk10 P_vc P_vb]));
%title(['Viterbi Probability  - 1:keg_6ft, 2:keg_8ft, 3:keg_10ft,
4:cylinder, test:background, M=' num2str(M) ', N=' num2str(N)])
if test==6
   xlabel('P_V_i_t_e_r_b_i[dB]')
end
if test==3
   ylabel('tested signal')
end

axis([-50 0 1 5]);
grid
hold
```

APPENDIX D. MATLAB CODE; NEURAL NETWORK BASED CLASSIFIER FOR MINE RECOGNITION

This Appendix contains MATLAB files used to recognize mine like objects using a supervised backpropagation feedforward neural network.

APPENDIX D1. NN TRAINING AND TESTING FOR MULTIPLE WEIGHTS SIGNAL MINE LIKE SIGNAL DATA

```
% Filename: nnlbs.m
% Written by: M. Zambartas
% Date Last Modified 10 August 1999
% Purpose: Performs mine like object signal classification - multiple
weights set-up
%          using a supervised backpropagation feedforward neural
network
% ts:          testing sequence used for the testing/training rotation
% O_multi_keg: multiple observation of keg signals
% O_multi_cyl: multiple observation of cylinder signals
% p:           input vectors
% t:           target vector; 1 for keg, 2 for cylinder
% sc:          output matrix

 seq=[2 3 4 5 6;1 3 4 5 6;1 2 4 5 6;1 2 3 5 6;1 2 3 4 6;1 2 3 4 5];

 sc=[];

 for test=1:6;


  ts=seq(test,:);


O_multi_keg=[class(:,1,ts(1))';class(:,1,ts(2))';class(:,1,ts(3))';clas
s(:,1,ts(4))';class(:,1,ts(5))';class(:,2,ts(1))';class(:,2,ts(2))';cla
ss(:,2,ts(3))';class(:,2,ts(4))';class(:,2,ts(5))';class(:,3,ts(1))';cl
ass(:,3,ts(2))';class(:,3,ts(3))';class(:,3,ts(4))';class(:,3,ts(5))']'
;
O_multi_cyl=[class(:,4,ts(1))';class(:,4,ts(2))';class(:,4,ts(3))';clas
s(:,4,ts(4))';class(:,4,ts(5))';class(:,5,ts(1))';class(:,5,ts(2))';cla
ss(:,5,ts(3))';class(:,5,ts(4))';class(:,5,ts(5))';class(:,6,ts(1))';cl
ass(:,6,ts(2))';class(:,6,ts(3))';class(:,6,ts(4))';class(:,6,ts(5))']'
;
% generation of input vectors (matrix) p:
p=[O_multi_keg O_multi_cyl];

t=[1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2];
% # of hidden layers:60.  # of output layers:1 Functions used: logsing,
purelin for hidden
% and output layer relatively
net=newff([1 8; 1 8],[60 1],{'logsig' 'purelin'},'trainlm');
figure(1)
net.performFcn='sse';
net.trainParam.min_grad=1e-20
net.trainParam.goal=0.01;
net.trainParam.show=10;
net.trainParam.epochs=300;
[net,tr]=train(net,p,t);
O1=class(:,1,test);
keg224=sim(net,O1)
O2=class(:,2,test);
keg432=sim(net,O2)
```

```
O3=class(:,3,test);
keg536=sim(net,O3)
O4=class(:,4,test);
cyl364=sim(net,O4)
O5=class(:,5,test);
cyl468=sim(net,O5)
O6=class(:,6,test);
cyl572=sim(net,O6)
O7=class(:,7,test);
back=sim(net,O7)
sc=[sc;
   keg224, keg432, keg536, cyl364, cyl468, cyl572, back];
figure(2)
%subplot(7,1,test), stem(1:7,sc), axis([1 7 1 2.5]),hold
end
% NN output plot:
subplot(3,3,1), plot(1:6, sc(:,1)), axis([1 6 0 4]),title
(['keg224lbs']),grid
subplot(3,3,2), plot(1:6, sc(:,2)), axis([1 6 0 4]),title
(['keg432lbs']),grid
subplot(3,3,3), plot(1:6, sc(:,3)), axis([1 6 0 4]),title
(['keg536lbs']),grid
subplot(3,3,4), plot(1:6, sc(:,4)), axis([1 6 0 4]),title
(['cyl364lbs']),grid
subplot(3,3,5), plot(1:6, sc(:,5)), axis([1 6 0 4]),title
(['cyl468lbs']),grid
subplot(3,3,6), plot(1:6, sc(:,6)), axis([1 6 0 4]),title
(['cyl572lbs']),grid
subplot(3,3,7), plot(1:6, sc(:,7)), axis([1 6 0 4]),title
(['background']),grid
```

APPENDIX D2. NN TRAINING AND TESTING FOR MULTIPLE DISTANCES SIGNAL MINE LIKE SIGNAL DATA

```
% Filename: nnft.m
% Written by: M. Zambartas
% Date Last Modified 10 August 1999
% Purpose: Performs mine like object signal classification - multiple
distances set-up
%          using a supervised backpropagation feedforward neural
network
% ts:           testing sequence used for the testing/training rotation
% O_multi_keg: multiple observation of keg signals
% O_multi_cyl: multiple observation of cylinder signals
% p:            input vectors
% t:            target vector; 1 for keg, 2 for cylinder
% sc:           output matrix

 seq=[2 3 4 5 6;1 3 4 5 6;1 2 4 5 6;1 2 3 5 6;1 2 3 4 6;1 2 3 4 5];
 sc=[];
 for test=1:6;
  ts=seq(test,:);
O_multi_keg=[class(:,1,ts(1))';class(:,1,ts(2))';class(:,1,ts(3))';clas
s(:,1,ts(4))';class(:,1,ts(5))';class(:,2,ts(1))';class(:,2,ts(2))';cla
ss(:,2,ts(3))';class(:,2,ts(4))';class(:,2,ts(5))';class(:,3,ts(1))';cl
ass(:,3,ts(2))';class(:,3,ts(3))';class(:,3,ts(4))';class(:,3,ts(5))']'
;
O_multi_cyl=[class(:,4,ts(1))';class(:,4,ts(2))';class(:,4,ts(3))';clas
s(:,4,ts(4))';class(:,4,ts(5))']';
% generation of input vectors (matrix) p:
p=[O_multi_keg O_multi_cyl];

  t=[1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2];
% # of hidden layers:60.  # of output layers:1 Functions used: logsing,
purelin for hidden
% and output layer relatively
net=newff([1 8; 1 8],[60 1],{'logsig' 'purelin'},'trainlm');
figure(1)
net.performFcn='sse';
net.trainParam.min_grad=1e-20
net.trainParam.goal=0.001;
net.trainParam.show=10;
net.trainParam.epochs=300;
[net,tr]=train(net,p,t);
O1=class(:,1,test);
keg6ft=sim(net,O1)
O2=class(:,2,test);
keg8ft=sim(net,O2)
O3=class(:,3,test);
keg10ft=sim(net,O3)
O4=class(:,4,test);
cyl10ft=sim(net,O4)
O5=class(:,5,test);
back=sim(net,O5)
sc=[sc;
   keg6ft, keg8ft, keg10ft, cyl10ft,back];
```

```
figure(2)

end
subplot(3,3,1), plot(1:6, sc(:,1),'o'), axis([1 6 0 4]),title
(['keg6ft']),grid
subplot(3,3,2), plot(1:6, sc(:,2)), axis([1 6 0 4]),title
(['keg8ft']),grid
subplot(3,3,3), plot(1:6, sc(:,3)), axis([1 6 0 4]),title
(['keg10ft']),grid
subplot(3,3,4), plot(1:6, sc(:,4)), axis([1 6 0 4]),title
(['cyl10ft']),grid
subplot(3,3,5), plot(1:6, sc(:,5)), axis([1 6 0 4]),title
(['background']),grid
```

LIST OF REFERENCES

[1] C. W. Therrien, *Discrete Random Signals and Statistical Signal Processing*, Prentice Hall, Englewood Cliffs, NJ, 1992.

[2] M. T. Hagan, Howard B. Demuth, and Mark Beale, *Neural Network Design*, PWS Publishing Co., MA, 1996.

[3] L.R. Rabiner and B.H. Juang, "An Introduction to Hidden Markov Models," IEEE ASSP magazine, Vol.3, No.1, pp. 4-16, 1986.

[4] L. R. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," Vol. 77, No. 2, pp. 257-284, 1989.

[5] S. K. Riis, "Hidden Markov Models and Neural Networks for Speech Recognition," http://eivind.imm.dtu.dk/staff/riis/riis.html, November 1998.

[6] D. H. Kil and F. B. Shin, *Pattern Recognition and Prediction with Applications to Signal Characterization*, American Institute of Physics, Woodbury, NY, 1996.

[7] R. M. Gray, "Vector Quantization," Vol. 1, No. 2, pp. 1-29, 1986.

[8] O. Cappe, Vector Quantization MATLAB files vq.mat, svq.mat, http://sig.enst.fr/~cappe/docs/hmmbib.html, November 1998.

[9] J. R. Deller, J. G. Proakis, and J. H. L. Hansen, *Discrete-Time Processing of Speech Signals*, Macmillan Publishing Company, NY, 1993.

[10] F. E. Gagham, *Discrete-Mode Source Development and Testing for New Seismo-Accoustic Sonar*, Master's Thesis, Naval Postgraduate School, Monterey, CA, March 1998.

[11] S. M. Fitzpatrick, *Source Development for a Seismo-Acoustic Sonar*, Master's Thesis, Naval Postgraduate School, Monterey, CA, December 1998.

[12] P. W. Hall, *Detection and Target-Strength Measurements of Buried Objects Using a Seismo-Acoustic Sonar*, Master's Thesis, Naval Postgraduate School, Monterey, CA, December 1998.

[13] C.M. Bishop, *Neural networks for pattern recognition*, Clarendon Press, 1997.

[14] Matlab Version 5.3.0.10183 (Neural Networks Toolbox), January 1999, The Mathworks Inc., Natick, MA.

[15] Prof. Muir, personal communication, September 1999.

INITIAL DISTRIBUTION LIST

No. Copies

1. Defense Technical Information Center.....2
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir VA 22060-6218

2. Engineering and Technology Curriculum Office, Code 34.....1
   Naval Postgraduate School
   Monterey, CA 93943-5109

3. Dudley Knox Library.....2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey CA 93943-5101

4. Chairman, Code EC.....1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, CA 93943-5121

5. Prof. Monique Fargues, CodeEC/Fa.....2
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, CA 93943-5121

6. Prof. Roberto Cristi, Code EC/Cx.....2
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, CA 93943-5121

7. Professor Thomas G. Muir, Code PH/Mt.....2
   Department of Physics
   Naval Postgraduate School
   Monterey, CA 93943-5117

8. LT Michail Zambartas.....3
   Amasias 18, Pagrati
   Athens-11634
   Greece